DDS Security

Version 1.1

IPR mode: *Non-Assert*

means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: http://www.omg.org/legal/tm_list.htm. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or

conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page: http://issues.omg.org/issues/create-new-issue.

Table of Contents

Tables

Figures

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling, and vertical domain frameworks. A listing of all OMG Specifications is available from the OMG website at:

http://www.omg.org/spec/index.htm

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

> OMG Headquarters
> 109 Highland Avenue
> Needham, MA 02494
> USA
> Tel: +1-781-444-0404
> Fax: +1-781-444-0320
> Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org.*

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to http://issues.omg.org/issues/create-new-issue.

1 Scope

1.1 General

This specification adds several new "DDS Security Support" compliance points ("profile") to the DDS Specification. See the compliance levels within the Conformance Clause below.

1.2 Overview of this Specification

This specification defines the Security Model and Service Plugin Interface (SPI) architecture for compliant DDS implementations. The DDS Security Model is enforced by the invocation of these SPIs by the DDS implementation. This specification also defines a set of builtin implementations of these SPIs.

- The specified builtin SPI implementations enable out-of-the box security and interoperability between compliant DDS applications.

- The use of SPIs allows DDS users to customize the behavior and technologies that the DDS implementations use for Information Assurance, specifically customization of Authentication, Access Control, Encryption, Message Authentication, Digital Signing, Logging and Data Tagging.
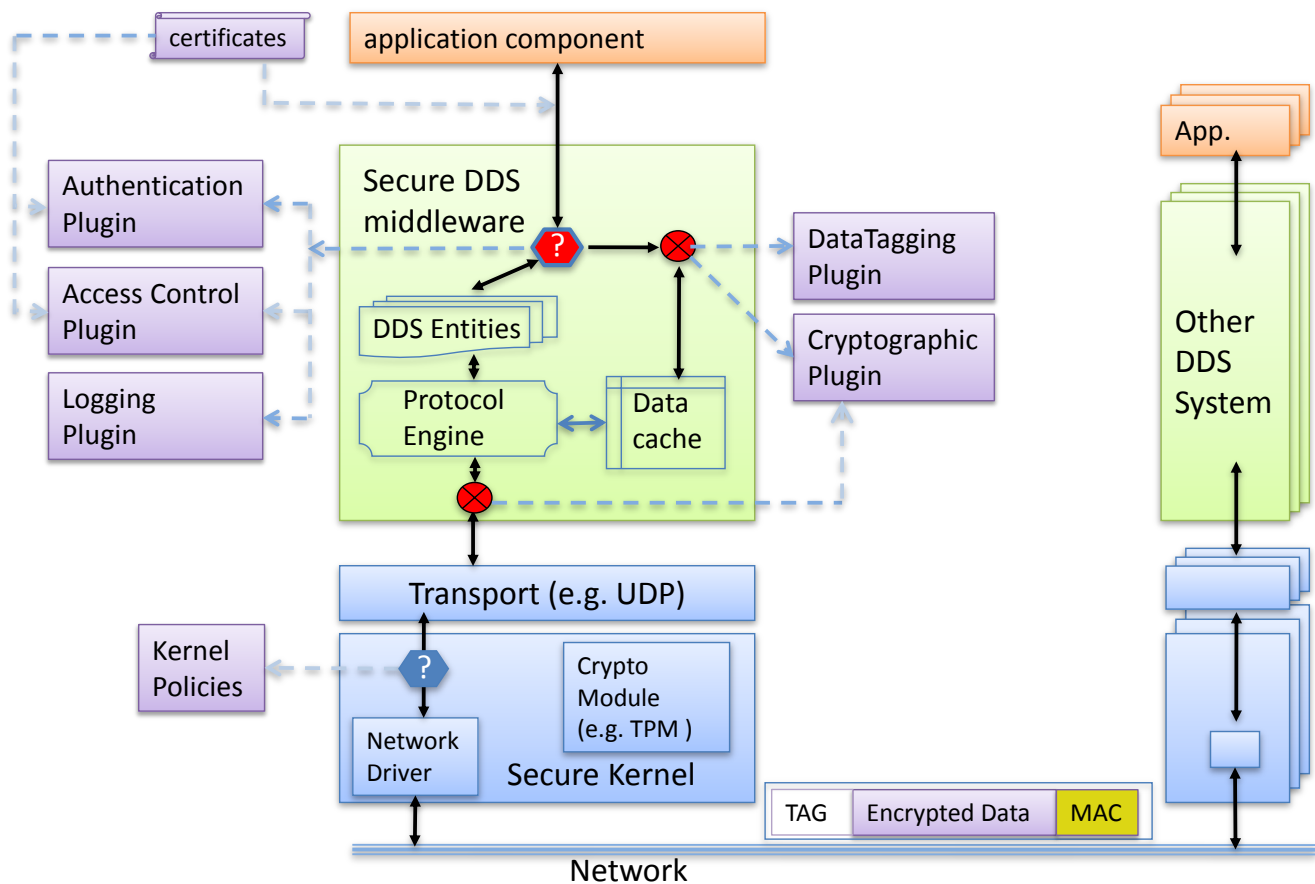
Figure 1 – Overall architecture for DDS Security

This specification defines five SPIs that when combined together provide Information Assurance to DDS systems:

- **Authentication** Service Plugin. Provides the means to verify the identity of the application and/or user that invokes operations on DDS. Includes facilities to perform mutual authentication between participants and establish a shared secret.

- **AccessControl** Service Plugin. Provides the means to enforce policy decisions on what DDS related operations an authenticated user can perform. For example, which domains it can join, which Topics it can publish or subscribe to, etc.

- **Cryptographic** Service Plugin. Implements (or interfaces with libraries that implement) all cryptographic operations including encryption, decryption, hashing, digital signatures, etc. This includes the means to derive keys from a shared secret.

- **Logging** Service Plugin. Supports auditing of all DDS security-relevant events

- **Data Tagging** Service Plugin. Provides a way to add tags to data samples.

2 Conformance

2.1 Conformance points

This specification defines the following conformance points:

(1) Builtin plugin interoperability (mandatory)
(2) Plugin framework (mandatory)
(3) Plugin language APIs (optional)
(4) Logging and Tagging (optional)

Conformance with the "DDS Security" specification requires conformance with all the mandatory conformance points.

2.2 Builtin plugin interoperability (mandatory)

This point provides interoperability with all the builtin plugins with the exception of the Logging plugin. Conformance to this point requires conformance to:

- Clause 7 (the security model and the support for interoperability between DDS Security implementations).
- The configuration of the plugins and the observable wire-protocol behavior specified in Clause 9 (the builtin-plugins), except for sub clause 9.6. This conformance point does not require implementation of the APIs between the DDS implementation and the plugins.

2.3 Plugin framework (mandatory):

This point provides the architectural framework and abstract APIs needed to develop new security plugins and "plug them" into a DDS middleware implementation. Plugins developed using this framework are portable between conforming DDS implementations. However portability for a specific programming language also requires conformance to the specific language API (see 2.4).

Conformance to this point requires conformance to:

- Clause 7 (the security model and the support for interoperability between DDS Security implementations).
- Clause 8 (the plugin model) with the exception of 8.6 and 8.7 (Logging and Data Tagging plugins). The conformance to the plugin model is at the UML level; it does not mandate a particular language mapping.
- Clause 9, the builtin-plugins, except for 9.6 (Builtin Logging Plugin).

In addition it requires the conforming DDS implementation to provide a public API to insert the plugins that conform to the aforementioned sections.

2.4 Plugin Language APIs (optional):

These conformance points provide portability across compliant DDS implementations of the security plugins developed using a specific programming language.

Conformance to any of the language portability points requires conformance to the (mandatory) plugin architecture framework point.

These are 5 "plugin language API" points, each corresponding to a different programming language used to implement the plugins.

Each language point is a separate independent conformance point. Conformance with the "plugin language API" point requires conformance with at least one of the 5 language APIs enumerated below:

- C Plugin APIs. Conformance to sub clauses 10.2 and 10.3
- C++ classic Plugin APIs. Conformance to sub clauses 10.2 and 10.4
- Java classic Plugin APIs. Conformance to sub clauses 10.2 and 10.5
- C++11 Plugin APIs. Conformance to sub clauses 10.2 and 10.6
- Java5+ Plugin APIs. Conformance to sub clauses 10.2 and 10.7.

2.5   Logging and Tagging profile (optional):

This point adds support for logging and tagging. Conformance to this point requires conformance to sub clauses 8.6, 8.7, and 9.6.

3 Normative References

- DDS: Data-Distribution Service for Real-Time Systems version 1.4.
  http://www.omg.org/spec/DDS/1.4
- DDS-RTPS: Data-Distribution Service Interoperability Wire Protocol version 2.2,
  http://www.omg.org/spec/DDS-RTPS/2.2/
- DDS-XTYPES: Extensible and Dynamic Topic-Types for DDS version 1.2,
  http://www.omg.org/spec/DDS-XTypes/1.2
- OMG-IDL: Interface Definition Language (IDL) version 4.1, http://www.omg.org/spec/IDL/4.1
- HMAC: Keyed-Hashing for Message Authentication. H. Krawczyk, M. Bellare, and R.Canetti,
  IETF RFC 2104, http://tools.ietf.org/html/rfc2104
- Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms,
  IETF RFC 6151 https://tools.ietf.org/html/rfc6151
- PKCS #7: Cryptographic Message Syntax Version 1.5. IETF RFC 2315.
  http://tools.ietf.org/html/rfc2315
- Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2.
  IETF RFC 8017. https://tools.ietf.org/html/rfc8017
- XSD: XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes,
  https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405.

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply:

Access Control

Mechanism that enables an authority to control access to areas and resources in a given physical facility or computer-based information system.

Authentication

Security measure(s) designed to establish the identity of a transmission, message, or originator.

Authorization

Access privileges that are granted to an entity; conveying an "official" sanction to perform a security function or activity.

Ciphertext

Data in its encrypted or signed form.

Certification authority

The entity in a Public Key Infrastructure (PKI) that is responsible for issuing certificates, and exacting compliance to a PKI policy.

Confidentiality

Assurance that information is not disclosed to unauthorized individuals, processes, or devices.

Cryptographic algorithm

A well-defined computational procedure that takes variable inputs, including a cryptographic key and produces an output.

Cryptographic key

A parameter used in conjunction with a cryptographic algorithm that operates in such a way that another agent with knowledge of the key can reproduce or reverse the operation, while an agent without knowledge of the key cannot.

Examples include:

1. The transformation of plaintext data into ciphertext.
2. The transformation of ciphertext data into plaintext.
3. The computation of a digital signature from data.
4. The verification of a digital signature.
5. The computation of a message authentication code from data.
6. The verification of a message authentication code from data and a received authentication code.

Data-Centric Publish-Subscribe (DCPS)

The mandatory portion of the DDS specification used to provide the functionality required for an application to publish and subscribe to the values of data objects.

Data Distribution Service (DDS)

An OMG distributed data communications specification that allows Quality of Service policies to be specified for data timeliness and reliability. It is independent of the implementation language.

Data Integrity

Assurance that data has not been altered since creation time.

Data-Origin Authentication

A mechanism providing assurance that a party is corroborated as the source of specified data (it includes data integrity). In this specification it is used to indicate assurance of the `DataWriter` or `DataReader` that originated a message.

Digital signature

The result of a cryptographic transformation of data that, when properly implemented with supporting infrastructure and policy, provides the services of:

1. origin authentication
2. data integrity
3. signer non-repudiation

Extended IDL

Extended Interface Definition Language (IDL) used to describe data types in a way that can be represented in a machine neutral format for network communications. This syntax was introduced as part of the DDS-XTYPES specification [3].

Hashing algorithm

A one-way algorithm that maps an input byte buffer of arbitrary length to an output fixed-length byte array in such a way that:

(a) Given the output it is computationally infeasible to determine the input.
(b) It is computationally infeasible to find any two distinct inputs that map to the same output.

Information Assurance

The practice of managing risks related to the use, processing, storage, and transmission of information or data and the systems and processes used for those purposes.

Integrity

Protection against unauthorized modification or destruction of information.

Key management

The handling of cryptographic material (e.g., keys, Initialization Vectors) during their entire life cycle of from creation to destruction.

Message authentication code (MAC)

A cryptographic hashing algorithm on data that uses a symmetric key to detect both accidental and intentional modifications of data.

Message-Origin Authentication

A mechanism providing assurance that a party is corroborated as the source of a specified message. In this specification it is used to indicate assurance of the `DomainParticipant` that originated the message.

Non-Repudiation

Assurance that the sender of data is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having received or processed the data.

Public key

A cryptographic key used with a public key cryptographic algorithm that is uniquely associated with an entity and that may be made public. The public key is associated with a private key. The public key may be known by anyone and, depending on the algorithm, may be used to:

1. Verify a digital signature that is signed by the corresponding private key,

2. Encrypt data that can be decrypted by the corresponding private key, or

3. Compute a piece of shared data.

Public key certificate

A set of data that uniquely identifies an entity, contains the entity's public key and possibly other information, and is digitally signed by a trusted party, thereby binding the public key to the entity.

Public key cryptographic algorithm

A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that determining the private key from the public key is computationally infeasible.

Public Key Infrastructure

A framework that is established to issue, maintain and revoke public key certificates.

5 Symbols

This specification does not define any symbols or abbreviations.

6 Additional Information

6.1 Changes to Adopted OMG Specifications

This specification does not modify any existing adopted OMG specifications. It reuses and/or adds functionality on top of the current set of OMG specifications.

- **DDS**: This specification does not modify or invalidate any existing DDS profiles or compliance levels. It extends some of the DDS builtin Topics to carry additional information in a compatible way with existing implementations of DDS.

- **DDS-RTPS**: This specification does not require any modifications to RTPS; however, it may impact interoperability with existing DDS-RTPS implementations. In particular, DDS-RTPS implementations that do *not* implement the DDS Security specification will have limited interoperability with implementations that *do* implement the mechanisms introduced by this specification. Interoperability is limited to systems configured to allow "unauthorized" DomainParticipant entities and within those systems, only to Topics configured to be "unprotected."

- **DDS-XTYPES**: This specification depends on the IDL syntax introduced by and the Extended CDR encoding defined in the DDS-XTYPES specification. It does not require any modifications of DDS-XTYPES.

- **OMG IDL**: This specification does not modify any existing IDL-related compliance levels.

6.2 Acknowledgments

The following individials and companies submitted content that was incorporated into this specification:

Submitting contributors:

- (lead) Gerardo Pardo-Castellote, Ph.D., Real-Time Innovations. gerardo.pardo AT rti.com

- Jaime Martin-Losa, eProsima JaimeMartin AT eprosima.com

- Angelo Corsaro, Ph.D., PrismTech. angelo.corsaro AT prismtech.com

Supporting contributors:

- Char Wales, MITRE charwing AT mitre.org

- Clark Tucker, Twin Oaks Computing, Inc. ctucker AT twinoakscomputing.com

Finalization Task Force members and participants:

- (chair) Gerardo Pardo-Castellote, Ph.D., Real-Time Innovations. gerardo.pardo AT rti.com

- Clark Tucker, Twin Oaks Computing, Inc. ctucker AT twinoakscomputing.com

- Jaime Martin-Losa, eProsima JaimeMartin AT eprosima.com

- Virginie Watine, THALES, virginie.watine AT thalesgroup.com

- Cyril Dangerville, THALES, cyril.dangerville AT thalesgroup.com

- Angelo Corsaro, Ph.D., PrismTech. angelo.corsaro AT prismtech.com

- Julien Enoch, PrismTech, julien.enoch AT prismtech.com

- Ricardo Gonzalez, eProsima, RicardoGonzalez AT eprosima.com

- Gilles Bessens, Kongsberg Gallium, gilles.bessens AT kongsberggallium.com

- Charles Fudge, NSWC Dalghren, charles.fudge AT navy.mil

- Ron Townsen, General Dynamics AIS, Ronald.Townsen AT gd-ais.com

Revision Task Force members and participants:

- (chair) Gerardo Pardo-Castellote, Ph.D., Real-Time Innovations. gerardo.pardo AT rti.com

- Clark Tucker, Twin Oaks Computing, Inc. ctucker AT twinoakscomputing.com

- Cyril Dangerville, THALES, cyril.dangerville AT thalesgroup.com

- Angelo Corsaro, Ph.D., PrismTech. angelo.corsaro AT prismtech.com

- Julien Enoch, PrismTech, julien.enoch AT prismtech.com

- Jose Maria Lopez-Vega, Ph.D., Real-Time Innovations. jose AT rti.com

- Yusheng Yang, Real-Time Innovations. yusheng AT rti.com

- Charles Fudge, NSWC Dalghren, charles.fudge AT navy.mil

- Ron Townsen, General Dynamics AIS, Ronald.Townsen AT gd-ais.com

7 Support for DDS Security

7.1 Security Model

The Security Model for DDS defines the security principals (users of the system), the objects that are being secured, and the operations on the objects that are to be restricted. DDS applications share information on DDS Global Data Spaces (called DDS Domains) where the information is organized into Topics and accessed by means of read and write operations on data-instances of those Topics.

Ultimately what is being secured is a specific DDS Global Data Space (domain) and, within the domain, the ability to access (read or write) information (specific Topic or even data-object instances within the Topic) in the DDS Global Data Space.

Securing DDS means providing:

- Confidentiality of the data samples

- Integrity of the data samples and the messages that contain them

- Authentication of DDS writers and readers

- Authorization of DDS writers and readers

- Message-origin authentication

- Data-origin authentication

- (Optional) Non-repudiation of data

To provide secure access to the DDS Global Data Space, applications that use DDS must first be authenticated, so that the identity of the application (and potentially the user that interacts with it) can be established. Once authentication has been obtained, the next step is to enforce access control decisions that determine whether the application is allowed to perform specific actions. Examples of actions are: joining a DDS Domain, defining a new Topic, reading or writing a specific DDS Topic, and even reading or writing specific Topic instances (as identified by the values of key fields in the data). Enforcement of access control shall be supported by cryptographic techniques so that information confidentiality and integrity can be maintained, which in turn requires an infrastructure to manage and distribute the necessary cryptographic keys.

7.1.1 Threats

In order to understand the decisions made in the design of the plugins, it is important to understand some of the specific threats impacting applications that use DDS and DDS Interoperability Wire Protocol (RTPS).

Most relevant are four categories of threats:

1. Unauthorized subscription

2. Unauthorized publication

3. Tampering and replay

4. Unauthorized access to data

These threats are described in the context of a hypothetical communication scenario with six actors all attached to the same network:

- **Alice**. A DDS DomainParticipant who is authorized to publish data on a Topic T.

- **Bob**. A DDS DomainParticipant who is authorized to subscribe to data on a Topic T.

- **Eve**. An eavesdropper. Someone who is **not authorized** to subscribe to data on Topic T. However Eve uses the fact that she is connected to the same network to try to see the data.

- **Trudy**. An intruder. A DomainParticipant who is **not authorized** to publish on Topic T. However, Trudy uses the fact that she is connected to the same network to try to send data.

- **Mallory**. A malicious DDS DomainParticipant. Mallory is authorized to subscribe to data on Topic T but she is **not authorized** to publish on Topic T. However, Mallory will try to use information gained by subscribing to the data to publish in the network and try to convince Bob that she is a legitimate publisher.

- **Trent**. A trusted service who needs to receive and send information on Topic T. For example, Trent can be a persistence service or a relay service. He is trusted to relay information without having malicious intent. However he is not trusted to see the content of the information.
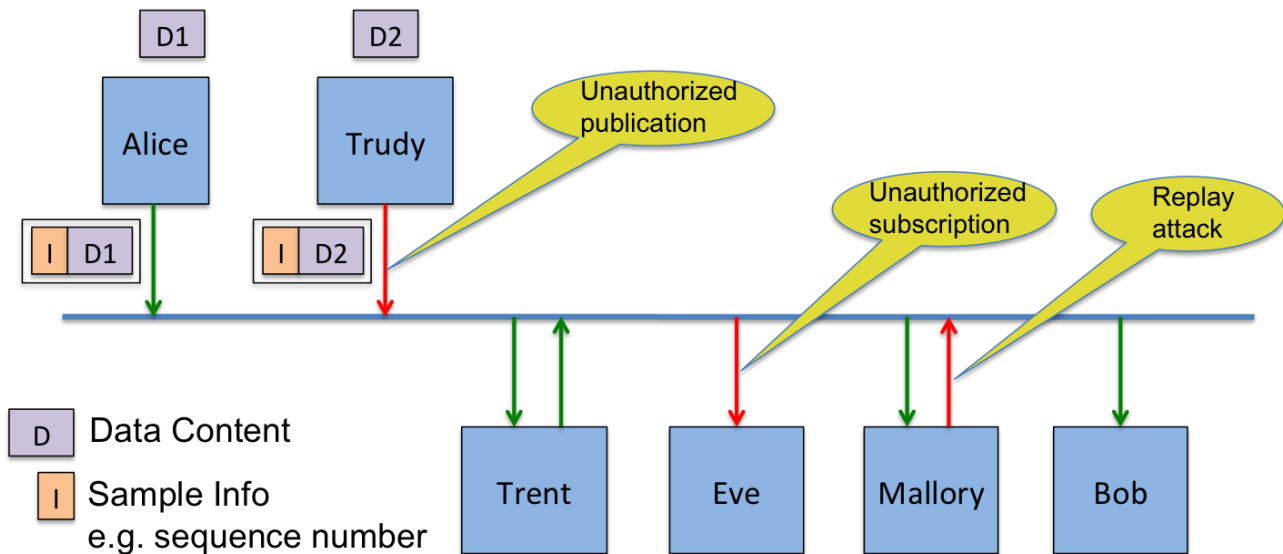
Figure 2 – Threat actors

7.1.1.1 Unauthorized Subscription

The DomainParticipant Eve is connected to the same network infrastructure as the rest of the agents and is able to observe the network packets despite the fact that the messages are not intended to be sent to Eve. Many scenarios can lead to this situation. Eve could tap into a network switch or observe the communication channels. Alternatively, in situations where Alice and Bob are communicating over multicast, Eve could simply subscribe to the same multicast address.

Protecting against Eve is reasonably simple. All that is required is for Alice to encrypt the data she writes using a secret key that is only shared with authorized receivers such as Bob, Trent, and Mallory.

7.1.1.2   Unauthorized Publication

The DomainParticipant Trudy is connected to the same network infrastructure as the rest of the agents and is able to inject network packets with any data contents, headers and destination she wishes (e.g., Bob). The network infrastructure will route those packets to the indicated destination.

To protect against Trudy, Bob, Trent and Mallory need to realize that the data is not originating from Alice. They need to realize that the data is coming from someone not authorized to send data on Topic T and therefore reject (i.e., not process) the packet.

Protecting against Trudy is also reasonably simple. All that is required is for the protocol to require that the messages include either a hash-based message authentication code (HMAC) or digital signature.

- An HMAC creates a message authentication code using a secret key that is shared with the intended recipients. Alice would only share the secret key with Bob, Mallory and Trent so that they can recognize messages that originate from Alice. Since Trudy is not authorized to publish Topic T, Bob and the others will not recognize any HMACs Trudy produces (i.e., they will not recognize Trudy's key).

- A digital signature is based on public key cryptography. To create a digital signature, Alice encrypts a digest of the message using Alice's private key. Everybody (including Bob, Mallory and Trent) has access to Alice's public key. Similar to the HMAC above, the recipients can identify messages from Alice, as they are the only ones whose digital signature can be interpreted with Alice's public key. Any digital signatures Trudy may use will be rejected by the recipients, as Trudy is not authorized to write Topic T.

The use of HMACs versus digital signatures presents tradeoffs that will be discussed further in subsequent sections. Suffice it to say that in many situations the use of HMACs is preferred because the performance to compute and verify them is about 1000 times faster than the performance of computing/verifying digital signatures.

7.1.1.3   Tampering and Replay

Mallory is authorized to subscribe to Topic T. Therefore Alice has shared with Mallory the secret key to encrypt the topic and also, if an HMAC is used, the secret key used for the HMAC.

Assume Alice used HMACs instead of digital signatures. Then Mallory can use her knowledge of the secret keys used for data encryption and the HMACs to create a message on the network and pretend it came from Alice. Mallory can fake all the TCP/UDP/IP headers and any necessary RTPS identifiers (e.g., Alice's RTPS DomainParticipant and DataWriter GUIDs). Mallory has the secret key that was used to encrypt the data so she can create encrypted data payloads with any contents she wants. She has the secret key used to compute HMACs so she can also create a valid HMAC for the new message. Bob and the others will have no way to see that message came from Mallory and will accept it, thinking it came from Alice.

So if Alice used an HMAC, the only solution to the problem is that the secret key used for the HMAC when sending the message to Mallory cannot be the same as the key used for the HMAC when sending messages to Bob. In other words, Alice must share a **different** secret key for the HMAC with each recipient. Then Mallory will not have the HMAC key that Bob expects from Alice and the messages from Mallory to Bob will not be misinterpreted as coming from Alice.

Recall that Alice needs to be able to use multicast to communicate efficiently with multiple receivers. Therefore, if Alice wants to send an HMAC with a different key for every receiver, the only solution is

to append multiple HMACs to the multicast message with some key-id that allows the recipient to select the correct HMAC to verify.

If Alice uses digital signatures to protect the integrity of the message, then this 'masquerading' problem does not arise and Alice can send the same digital signature to all recipients. This makes using multicast simpler. However, the performance penalty of using digital signatures is so high that in many situations it will be better to compute and send multiple HMACs as described earlier.

7.1.1.4   Unauthorized Access to Data by Infrastructure Services

Infrastructure services, such as the DDS Persistence Service or relay services need to be able to receive messages, verify their integrity, store them, and send them to other participants on behalf of the original application.

These services can be trusted not to be malicious; however, often it is not desirable to grant them the privileges they would need to understand the contents of the data. They are allowed to store and forward the data, but not to see inside the data.

Trent is an example of such a service. To support deployment of these types of services, the security model needs to support the concept of having a participant, such as Trent, who is allowed to receive, process, and relay RTPS messages, but is not allowed to see the contents of the data within the message. In other words, he can see the headers and sample information (writer GUID, sequence numbers, keyhash and such) but not the message contents.

To support services like Trent, Alice needs to accept Trent as a valid destination for her messages on topic T and share with Trent only the secret key used to compute the HMAC for Trent, but not the secret key used to encrypt the data itself. In addition, Bob, Mallory and others need to accept Trent as someone who is able to write on Topic T and relay messages from Alice. This means two things: (1) accept and interpret messages encrypted with Alice's secret key and (2) allow Trent to include in his sample information, the information he got from Alice (writer GUID, sequence number and anything else needed to properly process the relayed message).

Assume Alice used an HMAC in the message sent to Trent. Trent will have received from Alice the secret key needed to verify the HMAC properly. Trent will be able to store the messages, but lacking the secret key used for its encryption, will be unable to see the data. When he relays the message to Bob, he will include the information that indicates the message originated from Alice and produce an HMAC with its own secret HMAC key that was shared with Bob. Bob will receive the message, verify the HMAC and see it is a relayed message from Alice. Bob recognizes Trent is authorized to relay messages, so Bob will accept the sample information that relates to Alice and process the message as if it had originated with Alice. In particular, he will use Alice's secret key to decrypt the data.

If Alice had used digital signatures, Trent would have two choices. If the digital signature only covered the data and the sample information he needs to relay from Alice, Trent could simply relay the digital signature as well. Otherwise, Trent could strip out the digital signature and put in his own HMAC. Similar to before, Bob recognizes that Trent is allowed to relay messages from Alice and will be able to properly verify and process the message.

7.2   Types used by DDS Security

The DDS security specification includes extensions to the DDS Interoperability Wire Protocol (DDS-RTPS), as well as, new API-level functions in the form of Security Plugins. The types described in sub clause 7.2 are used in these extensions.

7.2.1 Property_t

Section 9.3.2 of the DDS-RTPS specification defines `Property_t` as a data type that holds a pair of strings. One string is considered the property "name" and the other is the property "value" associated with that name.

The DDS Security specification extends the DDS-RTPS definition of `Property_t` to contain the additional boolean attribute "propagate" used to indicate whether a property is intended for local use only or should be propagated by DDS discovery.

The DDS-Security specification uses `Property_t` sequences as a generic data type to configure the security plugins, pass metadata and provide an extensible mechanism for vendors to configure the behavior of their plugins without breaking portability or interoperability.

`Property_t` objects with names that start with the prefix "`dds.sec.`" are reserved by this specification, including future versions of this specification. Plugin implementers can also use this mechanism to pass metadata and configure the behavior of their plugins. In order to avoid collisions with the value of the "name" attribute, implementers shall use property names that start with a prefix to an ICANN domain name they own, in reverse order. For example, the prefix would be "com.acme." for plugins developed by a hypothetical vendor that owns the domain "acme.com".

The names and interpretation of the expected properties shall be specified by each plugin implementation.

Table 1 – Property_t class

Property_t	
Attributes	
name	String
value	String
propagate	Boolean

7.2.1.1   IDL Representation for Property_t

The `Property_t` type may be used for information exchange over the network. When a `Property_t` is sent over the network it shall be serialized using Extended CDR format according to the Extended IDL representation [3] below.

```
@extensibility(APPENDABLE)
struct Property_t {
   string name;
   string value;
    @non-serialized boolean propagate;
};
typedef sequence< Property_t > PropertySeq;
```

7.2.2 BinaryProperty_t

`BinaryProperty_t` is a data type that holds a string and an octet sequence. The string is considered the property "name" and the octet sequence the property "value" associated with that name.

Sequences of `BinaryProperty_t` are used as a generic data type to configure the plugins, pass metadata and provide an extensible mechanism for vendors to configure the behavior of their plugins without breaking portability or interoperability.

`BinaryProperty_t` also contains the boolean attribute "propagate". Similar to `Property_t` this attribute is used to indicate weather the corresponding binary property is intended for local use only or shall be propagated by DDS discovery.

`BinaryProperty_t` objects with a "name" attribute that start with the prefix "`dds.sec.`" are reserved by this specification, including future versions of this specification.

Plugin implementers may use this mechanism to pass metadata and configure the behavior of their plugins. In order to avoid collisions with the value of the "name", attribute implementers shall use property names that start with a prefix to an ICANN domain name they own, in reverse order. For example, the prefix would be "com.acme." for plugins developed by a hypothetical vendor that owns the domain "acme.com".

The valid values of the "name" attribute and the interpretation of the associated "value" shall be specified by each plugin implementation.

Table 2 – BinaryProperty_t class

BinaryProperty_t	
Attributes	
name	`String`
value	`OctetSeq`
propagate	`Boolean`

7.2.2.1 IDL Representation for BinaryProperty_t

The `BinaryProperty_t` type may be used for information exchange over the network. When a `BinaryProperty_t` is sent over the network, it shall be serialized using Extended CDR format according to the Extended IDL representation [3] below.

```
@extensibility(APPENDABLE)
struct BinaryProperty_t {
   string name;
   OctetSeq value;
    @non-serialized boolean propagate;
};
typedef sequence< BinaryProperty_t > BinaryPropertySeq;
```

7.2.3 DataHolder

`DataHolder` is a data type used to hold generic data. It contains various attributes used to store data of different types and formats. `DataHolder` appears as a building block for other types, such as `Token` and `GenericMessageData`.

Table 3 – DataHolder class

DataHolder	
Attributes	
class_id	String
properties	PropertySeq
binary_properties	BinaryPropertySeq

7.2.3.1 IDL representation for DataHolder

The `DataHolder` type may be used for information exchange over the network. When a `DataHolder` is sent over the network, it shall be serialized using Extended CDR format according to the Extended IDL representation [3] below.

```
@extensibility(APPENDABLE)
struct DataHolder {
    string              class_id;
    PropertySeq         properties;
    BinaryPropertySeq   binary_properties;

};

typedef sequence<DataHolder> DataHolderSeq;
```

7.2.4 Token

The `Token` class provides a generic mechanism to pass information between security plugins using DDS as the transport. `Token` objects are meant for transmission over the network using DDS either embedded within the builtin topics sent via DDS discovery or via special DDS Topic entities defined in this specification.

The `Token` class is structurally identical to the `DataHolder` class and therefore has the same structure for all plugin implementations. However, the contents and interpretation of the `Token` objects shall be specified by each plugin implementation.

There are multiple specializations of the `Token` class. They all share the same format, but are used for different purposes. This is modeled by defining specialized classes.
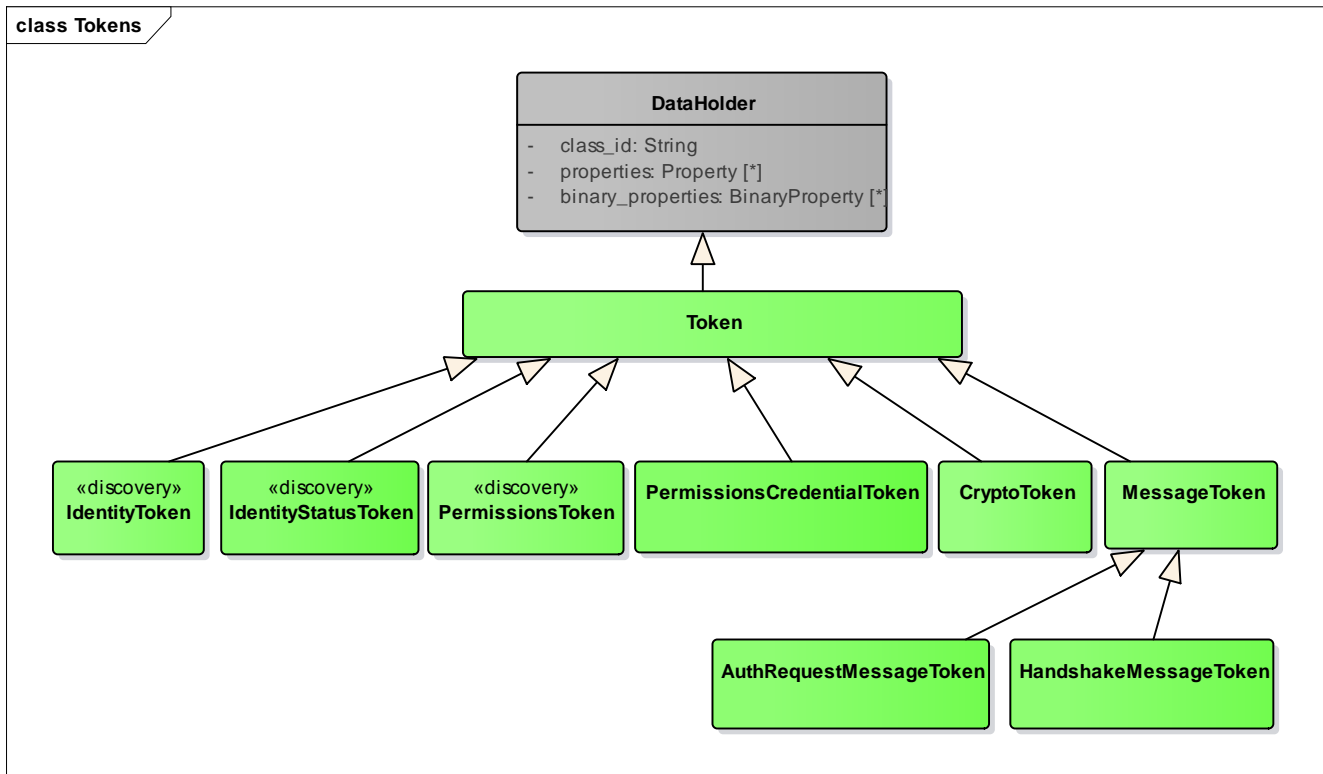
Figure 3 – Token Model

7.2.4.1   Attribute: `class_id`

When used as a `Token` class, the ***class_id*** attribute in the `DataHolder` identifies the kind of `Token`. Strings with the prefix "`dds.sec.`" are reserved for this specification, including future versions of the specification. Implementers of this specification can use this attribute to identify non-standard tokens. In order to avoid collisions, the ***class_id*** they use shall start with a prefix to an ICANN domain name they own, using the same rules specified in 7.2.1 for property names.

7.2.4.2   IDL Representation for Token and Specialized Classes

The `Token` class is used to hold information exchanged over the network. When a `Token` is sent over the network, it shall be serialized using Extended CDR format according to the Extended IDL representation below:

```
typedef DataHolder Token;

typedef Token MessageToken;
typedef MessageToken AuthRequestMessageToken;
typedef MessageToken HandshakeMessageToken;

typedef Token IdentityToken;
typedef Token IdentityStatusToken;
typedef Token PermissionsToken;
typedef Token AuthenticatedPeerCredentialToken;
typedef Token PermissionsCredentialToken;

typedef Token CryptoToken;
```

```
typedef Token ParticipantCryptoToken;
typedef Token DatawriterCryptoToken;
typedef Token DatareaderCryptoToken;

typedef sequence<HandshakeMessageToken> HandshakeMessageTokenSeq;
typedef sequence<CryptoToken> CryptoTokenSeq;
typedef CryptoTokenSeq ParticipantCryptoTokenSeq;
typedef CryptoTokenSeq DatawriterCryptoTokenSeq;
typedef CryptoTokenSeq DatareaderCryptoTokenSeq;
```

7.2.4.3 TokenNIL

This name refers to the `Token` object having *class_id* set to the empty string, and both *properties* and *binary_properties* sequences set to the empty sequence.

The `TokenNIL` object is used to indicate the absence of a `Token`.

7.2.5 PropertyQosPolicy, DomainParticipantQos, DataWriterQos, and DataReaderQos

This specification also introduces an additional Qos policy called `PropertyQosPolicy`, which is defined by the following extended IDL:

```
@extensibility(APPENDABLE)
struct PropertyQosPolicy {
   PropertySeq        value;
   BinaryPropertySeq  binary_value;
};
```

The `PropertyQosPolicy` applies to the following DDS entities: `DomainParticipant`, `DataWriter`, and `DataReader`. To allow configuration of this policy from the DDS API the DDS Security specification extends the definitions of the DDS defined types `DomainParticipantQos`, `DataWriterQos`, and `DataReaderQos` with the additional member "property" of type `PropertyQosPolicy` as indicated in the extended IDL snippets below.

This specification also introduces a Qos policy called `DataTagQosPolicy`, defined by the following IDL:

```
@extensibility(APPENDABLE)
struct Tag {
    string name;
    string value;
};

typedef sequence<Tag> TagSeq;

@extensibility(APPENDABLE)
struct DataTags {
    TagSeq tags;
};

typedef DataTags DataTagQosPolicy;
```

```
@extensibility(MUTABLE)
struct DomainParticipantQos {
   // Existing policies from the DDS specification
   PropertyQosPolicy  property;
};


@extensibility(MUTABLE)
struct DataWriterQos {
   // Existing policies from the DDS specification
   PropertyQosPolicy  property;
   DataTagQosPolicy   data_tags;
};

@extensibility(MUTABLE)
struct DataReaderQos {
   // Existing policies from the DDS specification
   PropertyQosPolicy  property;
   DataTagQosPolicy   data_tags;
};
```

The `PropertyQosPolicy` shall be propagated via DDS discovery so it appears in the
`ParticipantBuiltinTopicData`, `PublicationBuiltinTopicData`, and
`SubscriptionBuiltinTopicData` (see 7.4.1.3, 7.4.1.7, and 7.4.1.8). This is used by the plugins
to check configuration compatibility. Not all name/value pairs within the underlying `PropertySeq`
and `BinaryPropertySeq` are propagated. Specifically only the ones with propagate=TRUE are
propagated via DDS discovery and shall appear in the `ParticipantBuiltinTopicData`,
`PublicationBuiltinTopicData`, and `SubscriptionBuiltinTopicData`.

7.2.6 ParticipantGenericMessage

This specification introduces additional builtin `DataWriter` and `DataReader` entities used to send
generic messages between the participants. To support these entities, this specification uses a general-
purpose data type called `ParticipantGenericMessage`, which is defined by the following
extended IDL:

```
typedef octet[16] GUID_t;

@extensibility(APPENDABLE)
struct MessageIdentity {
   GUID_t source_guid;
   long long sequence_number;
};


typedef string<> GenericMessageClassId;

@extensibility(APPENDABLE)
struct ParticipantGenericMessage {
```

```
    /* target for the request. Can be GUID_UNKNOWN */
  MessageIdentity message_identity;
  MessageIdentity related_message_identity;
  GUID_t destination_participant_guid;
  GUID_t destination_endpoint_guid;
  GUID_t source_endpoint_guid;
  GenericMessageClassId   message_class_id;
  DataHolderSeq           message_data;
};
```

The type `GUID_t` refers to the type defined in the DDS-RTPS specification [2]. See clause 7.3.3 for additional details on the `GUID_t`.

7.2.7 ParticipantSecurityInfo

This specification introduces a new set of participant security attributes, described in Section 8.4.2.4.

In order to communicate, two participants need to have a compatible configuration for participant security attributes. To support making matching decisions upon discovering a remote participant, this specification defines a new parameter ID for ***ParticipantBuiltinTopicData*** topic, PID_PARTICIPANT_SECURITY_INFO (see Section 7.4.1.4). The type for that Parameter IDs is defined by the following extended IDL:

```
typedef unsigned long ParticipantSecurityAttributesMask;

typedef unsigned long PluginParticicipantSecurityAttributesMask;


@extensibility (APPENDABLE)

struct ParticipantSecurityInfo {

  ParticipantSecurityAttributesMask       participant_security_atributes;

  PluginParticipantSecurityAttributesMask plugin_participant_security_attributes;

};
```

```
#define  PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_VALID (0x1 << 31)
```

The default value for the info `ParticipantSecurityInfo` sets both masks to zero:

```
#define  PARTICIPANT_SECURITY_ATTRIBUTES_INFO_DEFAULT {0, 0}
```

A compatible configuration is defined as having the same value for all of the attributes in the `ParticipantSecurityInfo`, except that when comparing two masks the most significant bit is interpreted in a special manner as described below.

The most-significant bit of `PluginParticipantSecurityAttributesMask` and `ParticipantSecurityAttributesMask` is called the ***is_valid*** bit and specifies whether the rest of the mask is valid. If the ***is_valid*** is set to zero on either of the masks, the comparison between the local and remote setting for the `ParticipantSecurityInfo` shall ignore the attribute. This allows new implementations to be backwards compatible with old implementations by either not sending the `ParticipantSecurityInfo` (the default value of zero has ***is_valid***=0) or sending it with ***is_valid***.

The value of the ***plugin_participant_security_attributes*** shall be defined the security plugin implementation, and are opaque to the DDS middleware (other than the ***is_valid*** bit). They allow the middleware to make matching decisions using the `PluginParticipantSecurityAttributesMask` without interpreting it. The definition for the builtin plugins is detailed in clause 9.4.2.3.

7.2.8 EndpointSecurityInfo

This specification defines a plugin-independent endpoint security attributes, described in clause 8.4.2.7. Additionally, plugin implementations can also have their own plugin-specific attributes, see clause 9.4.2.5

In order to communicate, two endpoints need to have a compatible configuration for endpoint security attributes.

To support making matching decisions upon discovering a remote endpoint, this specification defines a new parameter ID for ***PublicationBuiltinTopicData*** and ***SubscriptionBuiltinTopicData*** topics, `PID_ENDPOINT_SECURITY_INFO` (see Section 7.4.1.5). The type for that Parameter IDs is defined by the following extended IDL:

```
typedef unsigned long EndpointSecurityAttributesMask;
typedef unsigned long PluginEndpointSecurityAttributesMask;

@extensibility(APPENDABLE)
struct EndpointSecurityInfo {
  EndpointSecurityAttributesMask       endpoint_security_attributes;
  PluginEndpointSecurityAttributesMask plugin_endpoint_security_attributes;
};
```

```
#define  ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_VALID (0x1 << 31)
```

The default value for the `EndpointSecurityInfo` is both attributes set to the value zero.

```
#define  ENDPOINT_SECURITY_ATTRIBUTES_INFO_DEFAULT {0, 0}
```

A compatible configuration is defined as having the same value for all of the attributes in the `EndpointSecurityInfo`, except that when comparing two masks the most significant bit is interpreted in a special manner as described below.

The most-significant bit of `PluginEndpointSecurityAttributesMask` and `EndpointSecurityAttributesMask` is called the ***is_valid*** bit and specifies whether the rest of the mask is valid. If the ***is_valid*** is set to zero on either of the masks, the comparison between the local and remote setting for the `EndpointSecurityInfo` shall ignore the attribute. This allows new implementations to be backwards compatible with old implementations by either not sending the `EndpointSecurityInfo` (the default value of zero has ***is_valid***=0) or sending it with ***is_valid*** bit set to zero in one or both attributes.

The value of the ***plugin_endpoint_security_attributes*** shall be defined by the security plugin implementation, and is opaque to the DDS middleware (other than the ***is_valid*** bit). It allows the middleware to make matching decisions using the

`PluginEndpointSecurityAttributesMask` without interpreting it. The definition for the builtin plugins is detailed in clause 9.4.2.5.

7.2.9 Additional DDS Return Code: NOT_ALLOWED_BY_SECURITY

The DDS specification defines a set of return codes that may be returned by the operations on the DDS API (see sub clause 7.1.1 of the DDS specification).

The DDS Security specification add an additional return code NOT_ALLOWED_BY_SECURITY, which shall be returned by any operation on the DDS API that fails because the security plugins do not allow it.

7.3   Securing DDS Messages on the Wire

OMG DDS uses the Real-Time Publish-Subscribe (RTPS) on-the-wire protocol [2] for communicating data. The RTPS protocol includes specifications on how discovery is performed, the metadata sent during discovery, and all the protocol messages and handshakes required to ensure reliability. RTPS also specifies how messages are put together.

7.3.1 RTPS Background (Non-Normative)

In a secure system where efficiency and message latency are also considerations, it is necessary to define exactly what needs to be secured. Some applications may require only the data payload to be confidential and it is acceptable for the discovery information, as well as, the reliability meta-traffic (HEARTBEATs, ACKs, NACKs, etc.) to be visible, as long as it is protected from modification. Other applications may also want to keep the metadata (sequence numbers, in-line QoS) and/or the reliability traffic (ACKs, NACKs, HEARTBEATs) confidential. In some cases, the discovery information (who is publishing what and its QoS) may need to be kept confidential as well.

To help clarify these requirements, sub clause 7.3.1 explains the structure of the RTPS `Message` and the different `Submessages` it may contain.
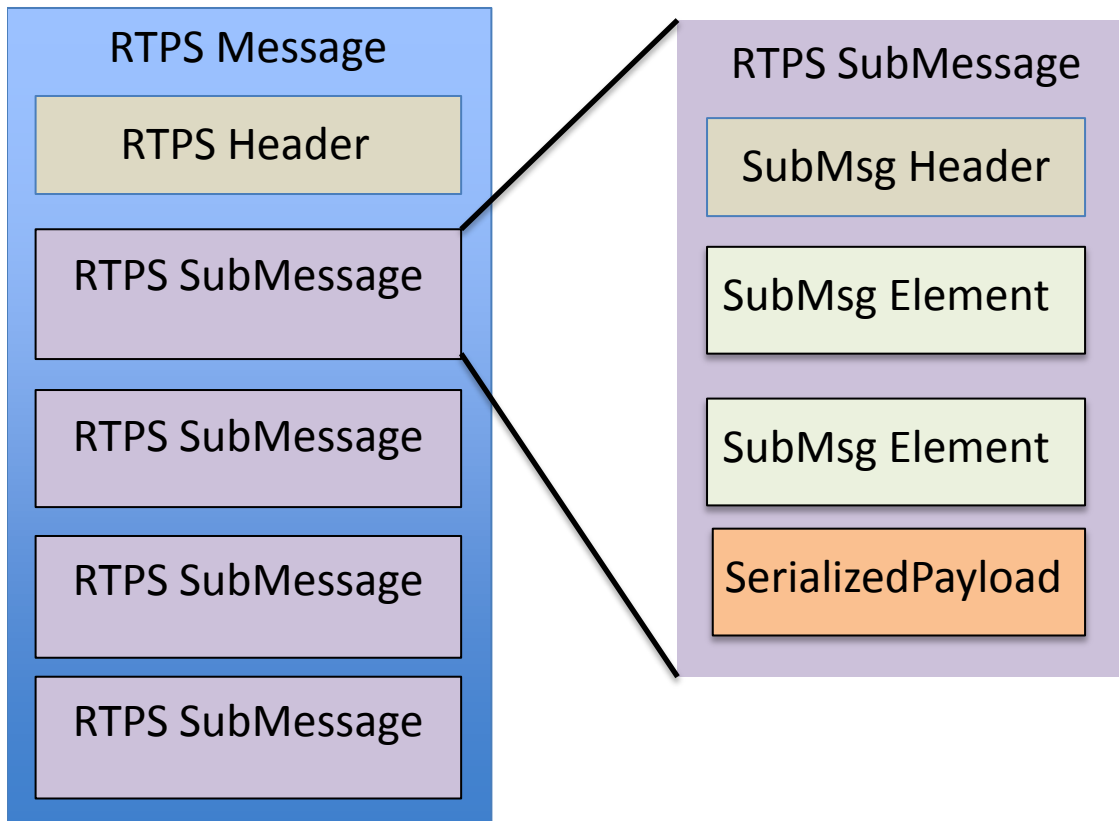
Figure 4 – RTPS message structure

An RTPS Message is composed of a leading RTPS `Header` followed by a variable number of RTPS `Submessages`. Each RTPS `Submessage` is composed of a `SubmessageHeader` followed by a variable number of `SubmessagElements`. There are various kinds of `SubmessageElements` to communicate things like sequence numbers, unique identifiers for DataReader and DataWriter entities, SerializedKeys or KeyHash of the application data, source timestamps, QoS, etc. There is one kind of `SubmessageElement` called `SerializedPayload` that is used to carry the data sent by DDS applications.

For the purposes of securing communications we distinguish three types of RTPS `Submessages`:

1. **DataWriter Submessages**. These are the RTPS submessages sent by a DataWriter to one or more DataReader entities. These include the `Data`, `DataFrag`, `Gap`, `Heartbeat`, and `HeartbeatFrag` submessages.
2. **DataReader Submessages.** These are the RTPS submessages sent by a DataReader to one or more DataWriter entities. These include the `AckNack` and `NackFrag` submessages.
3. **Interpreter Submessages.** These are the RTPS submessages that are destined to the Message Interpreter and affect the interpretation of subsequent submessages. These include all the "`Info`" messages.

The only RTPS submessages that contain application data are the `Data` and `DataFrag`. The application data is contained within the `SerializedPayload` submessage element. In addition to the `SerializedPayload` these submessages contain sequence numbers, inline QoS, the Key Hash, identifiers of the originating DataWriter and destination DataReader, etc.

The `Data`, and `DataFrag` submessages contain a `ParameterList` submessage element called *inlineQos* (see section 8.3.7 of the DDS-RTPS specification version 2.2). The *inlineQos* holds metadata associated with the submessage. It is encoded as a `ParameterList` (see section 9.4.2.11 of the DDS-RTPS specification version 2.2). `ParameterList` is a list of {paramaterId, length, value} tuples terminated by a sentinel. One of these parameters is the `KeyHash`.

The `KeyHash` parameter may only appear in the `Data` and `DataFrag` submessages. Depending on the data type associated with the DataWriter that wrote the data, the `KeyHash` parameter contains either:

- A serialized representation of the values of all the attributes declared as 'key' attributes in the associated data type, or
- An MD5 hash computed over the aforementioned serialized key attributes.

Different RTPS `Submessage` within the same RTPS `Message` may originate on different DataWriter or DataReader entities within the `DomainParticipant` that sent the RTPS message.

It is also possible for a single RTPS `Message` to combine submessages that originated on different DDS DomainParticipant entities. This is done by preceding the set of RTPS `Submessages` that originate from a common `DomainParticipant` with an `InfoSource` RTPS submessage.

The RTPS header contains the version of the RTPS protocol composed of a Major Version and Minor Version numbers.

As specified in clause 8.6.1 of the DDS-RTPS specification, changes to the RTPS protocol that do not break interoperability should increase the Minor Version number. These changes include additional submessages, additional builtin-endpoints, and additional parameterIds. The DDS Security specification makes these kinds of changes to the RTPS protocol and therefore must increase the RTPS minor version number.

7.3.2 Secure RTPS Messages

Sub clause 7.1.1 identified the threats addressed by the DDS Security specification. To protect against the "Unauthorized Subscription" threat it is necessary to use encryption to protect the sensitive parts of the RTPS message.

Depending on the application requirements, it may be that the only thing that should be kept confidential is the content of the application data; that is, the information contained in the `SerializedPayload` RTPS submessage element. However, other applications may also consider the information in other RTPS `SubmessageElements` (e.g., sequence numbers, `KeyHash`, and unique writer/reader identifiers) to be confidential. So the entire `Data` (or `DataFrag`) submessage may need to be encrypted. Similarly, certain applications may consider other submessages such as `Gap`, `AckNack`, `Heartbeat`, `HeartbeatFrag`, etc. also to be confidential.

For example, a `Gap` RTPS `Submessage` instructs a `DataReader` that a range of sequence numbers is no longer relevant. If an attacker can modify or forge a `Gap` message from a `DataWriter`, it can trick the `DataReader` into ignoring the data that the `DataWriter` is sending.

To protect against "Unauthorized Publication" and "Tampering and Replay" threats, messages must be signed using secure hashes or digital signatures. Depending on the application, it may be sufficient to sign only the application data (`SerializedPayload` submessage element), the whole `Submessage`, and/or the whole RTPS `Message`.

To support different deployment scenarios, this specification uses a "message transformation" mechanism that gives the Security Plugin Implementations fine-grain control over which parts of the RTPS `Message` need to be encrypted and/or signed.

The Message Transformation performed by the Security Plugins transforms an RTPS `Message` into another RTPS `Message`. A new RTPS `Header` may be added and the content of the original RTPS Message may be encrypted, protected by a Secure Message Authentication Code (MAC), and/or signed. The MAC and/or signature can also include the RTPS Header to protect its integrity.

7.3.3 Constraints of the DomainParticipant GUID_t (GUID)

The DDS-RTPS specification [2] states that DDS `DomainParticipant` entities are identified by a unique 16-byte GUID with type `GUID_t`. In this DDS-Security specification the type `GUID_t` refers to the same type defined in clauses 8.4.2.1 and 9.3.1 of the DDS-RTPS specification [2]:

```
// From DDS-RTPS [2] clauses 8.4.2.1 and 9.3.1

typedef octet GuidPrefix_t[12];

struct EntityId_t {

    octet entityKey[3];

    octet entityKind;

};

struct GUID_t {

    GuidPrefix_t prefix;

    EntityId_t   entityId;

};
```

This `DomainParticipant` GUID is communicated as part of DDS Discovery in the **SPDPdiscoveredParticipantData** (see DDS-RTPS specification [2] clauses 8.5.3.2 and 9.3.1.3).

Allowing a `DomainParticipant` to select its GUID arbitrarily would allow hostile applications to perform a "squatter" attack, whereby a `DomainParticipant` with a valid certificate could announce itself into the DDS Domain with the GUID of some other `DomainParticipant`. Once authenticated the "squatter" `DomainParticipant` would preclude the real DomainParticipant from being discovered, because its GUID would be detected as a duplicate of the already existing one.

To prevent the aforementioned "squatter" attack, this specification constrains the GUID that can be chosen by a `DomainParticipant`, so that it is tied to the Identity of the `DomainParticipant`. This is enforced by the `Authentication` plugin.

7.3.4 Mandatory use of the KeyHash for encrypted messages

The RTPS `Data` and `DataFrag` submessages can optionally contain the `KeyHash` as an inline Qos (see sub clause 9.6.3.3, titled "KeyHash (PID_KEY_HASH)") of the DDS-RTPS specification version 2.3. In this sub clause it is specified that when present, the key hash shall be computed either as the serialized key or as an MD5 on the serialized key.

The key values are logically part of the data and therefore in situations where the data is considered sensitive the key should also be considered sensitive.

For this reason the DDS Security specification imposes additional constraints in the use of the key hash. These constraints apply only to the `Data` or `DataFrag` RTPS `SubMessages` where the `SerializedPayload SubmessageElement` is encrypted by the operation `encode_serialized_payload` of the `CryptoTransform` plugin:

(1) The `KeyHash` shall be included in the Inline Qos.
(2) The `KeyHash` shall be computed as the 128 bit MD5 Digest (IETF RFC 1321) applied to the CDR Big- Endian encapsulation of all the Key fields in sequence. Unlike the rule stated in sub clause 9.6.3.3 of the DDS specification, the MD5 hash shall be used regardless of the maximum-size of the serialized key.

These rules accomplish two objectives:

(1) Avoid leaking the value of the key fields in situations where the data is considered sensitive and therefore appears encrypted within the Data or DataFrag submessages.
(2) Enable the operation of infrastructure services without needed to leak to them the value of the key fields (see 7.1.1.4).

Note that the use of the MD5 hashing function for these purposes does not introduce significant vulnerabilities. While MD5 is considered broken as far as resistance to collisions (being able to find two inputs that result in an identical unspecified hash) there are still no known practical preimage attacks on MD5 (being able to find the input that resulted on a given hash).

7.3.5 Immutability of Publisher Partition Qos in combination with non-volatile Durability kind

The DDS specification allows the `PartitionQos` policy of a `Publisher` to be changed after the `Publisher` has been enabled. See sub clause 7.1.3 titled "Supported QoS) of the DDS 1.2 specification.

The DDS Security specification restricts this situation.

The DDS implementation shall not allow a `Publisher` to change `PartitionQos` policy after the `Publisher` has been enabled if it contains any `DataWriter` that meets 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 `DurabilityQos` policy kind set to something other than `VOLATILE`.

This rule prevents data that was published while the `DataWriter` had associated a set of `Partitions` from being sent to `DataReaders` that were not matching before the `Partition` change and match after the `Partition` is changed.

7.3.6 Platform Independent Description

7.3.6.1 Change to the RTPS minor version number

Implementations of this specification shall set the RTPS protocol version number present in the RTPS Header. The RTPS Major version number shall be set to 2 and the RTPS Minor version number shall be set to 3. Future revisions of the DDS-RTPS specification shall take this fact into consideration.

7.3.6.2 RTPS Secure Submessage Elements

This specification introduces new RTPS `SubmessageElements` that may appear inside RTPS `Submessages`.

7.3.6.2.1 CryptoTransformIdentifier

The `CryptoTransformIdentifier` submessage element identifies the kind of cryptographic transformation that was performed in an RTPS `Submessage` or an RTPS `SubmessageElement` and also provides a unique identifier of the key material used for the cryptographic transformation.

The way in which attributes in the `CryptoTransformIdentifier` are set shall be specified for each Cryptographic plugin implementation. However, all Cryptographic plugin implementations shall be set in a way that allows the operations `preprocess_secure_submsg`, `decode_datareader_submessage`, `decode_datawriter_submessage`, and `decode_serialized_payload` to uniquely recognize the cryptographic material they shall use to decode the message, or recognize that they do not have the necessary key material.

7.3.6.2.2 CryptoContent

The `CryptoContent` submessage element is used to wrap a `SerializedPayload`, an RTPS `Submessage`, or a complete RTPS `Message`. It is the result of applying one of the encoding transformations on the `CryptoTransform` plugin.

The specific format of this shall be defined by each `Cryptographic` plugin implementation.

7.3.6.2.3 CryptoHeader

The `CryptoHeader` submessage element is used as prefix to wrap a `SerializedPayload`, an RTPS `Submessage`, or a complete RTPS `Message`. It is the result of applying one of the encoding transformations on the `CryptoTransform` plugin.

The `CryptoHeader` submessage element shall extend the `CryptoTransformIdentifier` element. Consequently, the leading bytes in the `CryptoHeader` shall encode the `CryptoTransformIdentifier`. Therefore, the **_transformationKind_** is guaranteed to be the first element within the `CryptoHeader`. The specific format of this shall be defined by each `Cryptographic` plugin implementation.

7.3.6.2.4 CryptoFooter

The `CryptoFooter` submessage element is used as postfix to wrap a `SerializedPayload`, an RTPS Submessage, or a complete RTPS Message. It is the result of applying one of the encoding transformations on the `CryptoTransform` plugin.

The specific format of this shall be defined by each `Cryptographic` plugin implementation.

7.3.6.3   RTPS Submessage: SecureBodySubMsg

This specification introduces a new RTPS submessage: `SecureBodySubMsg`. The format of the `SecureBodySubMsg` complies with the RTPS `SubMessage` format mandated in the RTPS specification. It consists of the RTPS `SubmessageHeader` followed by a set of RTPS `SubmessageElement` elements.

Since the `SecureBodySubMsg` conforms to the general structure of RTPS submessages, it can appear inside a well-formed RTPS message.

class SecureSubmessages
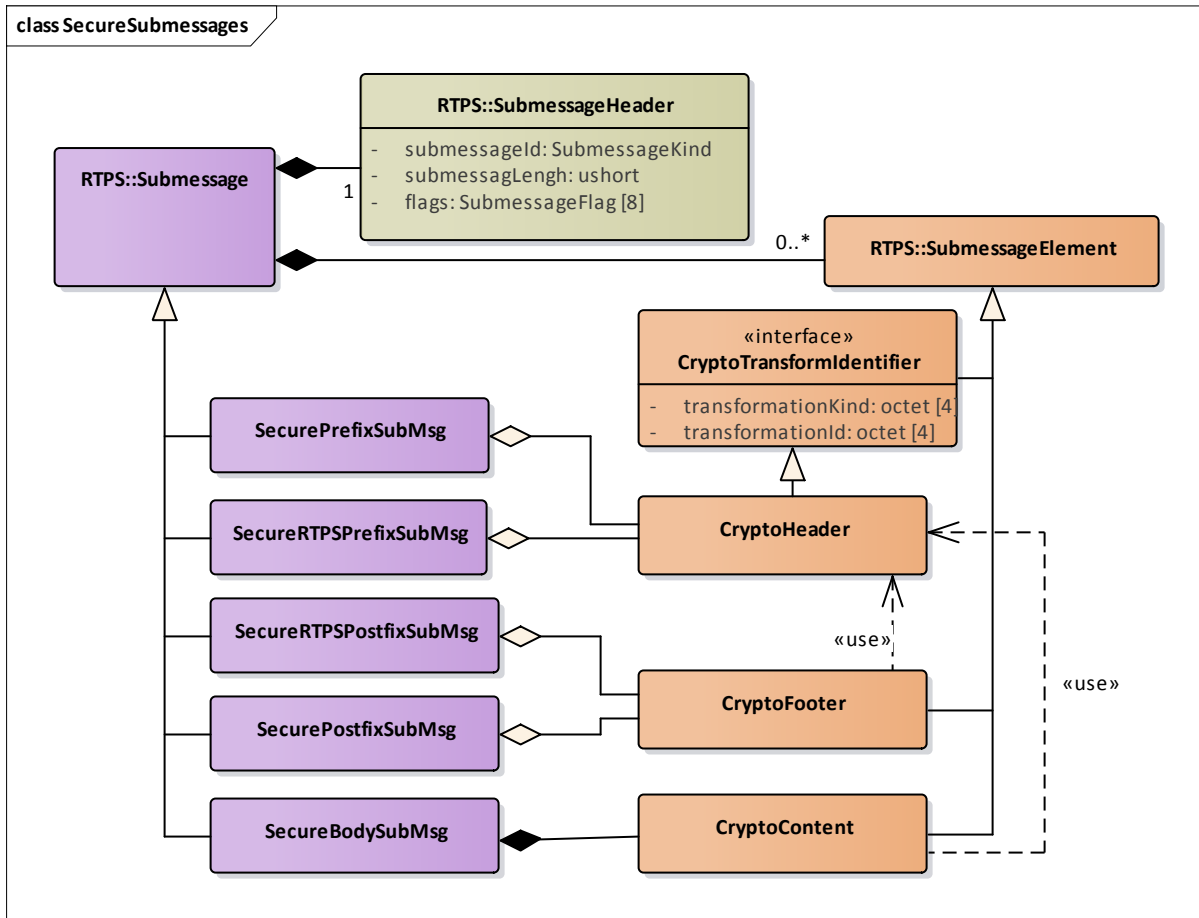
Figure 5 – Secure Submessage and Secured Payload Model

7.3.6.3.1   Purpose

The `SecureBodySubMsg` submessage is used to wrap one or more regular RTPS submessages in such a way that their contents are secured via encryption, message authentication, and/or digital signatures.

7.3.6.3.2   Content

The elements that form the structure of the RTPS `SecureBodySubMsg` are described in the table below.

Table 4 – SecureBodySubMsg class

Element	Type	Meaning
SEC_BODY	SubmessageKind	The presence of this field is common to RTPS submessages. It identifies the kind of submessage. The value indicates it is a SecureBodySubMsg
submessageLength	ushort	The presence of this field is common to RTPS submessages. It identifies the length of the submessage.
EndianessFlag	SubmessageFlag	Appears in the Submessage header flags. Indicates endianess.
crypto_content	CryptoContent	Contains the result of transforming the original message. Depending on the plugin implementation and configuration, it may contain encrypted content, message access codes, and/or digital signatures

7.3.6.3.3   Validity

The RTPS `Submessage` is invalid if the ***submessageLength*** in the Submessage header is too small.

7.3.6.3.4   Logical Interpretation

The `SecureBodySubMsg` provides a way to secure content inside a legal RTPS submessage.

A `SecureBodySubMsg` may wrap a single RTPS `Submessage` or a whole RTPS `Message`.

7.3.6.4   RTPS Submessage: SecurePrefixSubMsg

This specification introduces the RTPS submessage: `SecurePrefixSubMsg`. The format of the `SecurePrefixSubMsg` complies with the RTPS `SubMessage` format mandated in the RTPS specification. It consists of the RTPS `SubmessageHeader` followed by a set of RTPS `SubmessageElement` elements.

7.3.6.4.1   Purpose

The `SecurePrefixSubMsg` submessage is used as prefix to wrap an RTPS submessage in such a way that its contents are secured via encryption, message authentication, and/or digital signatures.

7.3.6.4.2   Content

The elements that form the structure of the RTPS `SecurePrefixSubMsg` are described in the table below.

Table 5 – SecurePrefixSubMsg class

Element	Type	Meaning
SEC_PREFIX	SubmessageKind	The presence of this field is common to RTPS submessages. It identifies the kind of submessage. The value indicates it is a SecurePrefixSubMsg
submessageLength	ushort	The presence of this field is common to RTPS submessages. It identifies the length of the submessage.
EndianessFlag	SubmessageFlag	Appears in the Submessage header flags. Indicates endianess.
transformation_id	CryptoTransformIdentifier	Identfies the kind of transformation performed on the RTPS Sububmessage that follows it.
plugin_crypto_header_extra	octet[]	Provides further information on the transformation performed. The contents are specific to the Plugin Implementation and the value of the transformation_id

7.3.6.4.3 Validity

The RTPS `Submessage` is invalid if the *submessageLength* in the Submessage header is too small.

7.3.6.4.4 Logical Interpretation

The `SecurePrefixSubMsg` provides a way to prefix secure content inside a legal RTPS submessage.

A `SecurePrefixSubMsg` shall be followed by a single RTPS `Submessage` which itself shall be followed by a `SecurePostfixSubMsg`.
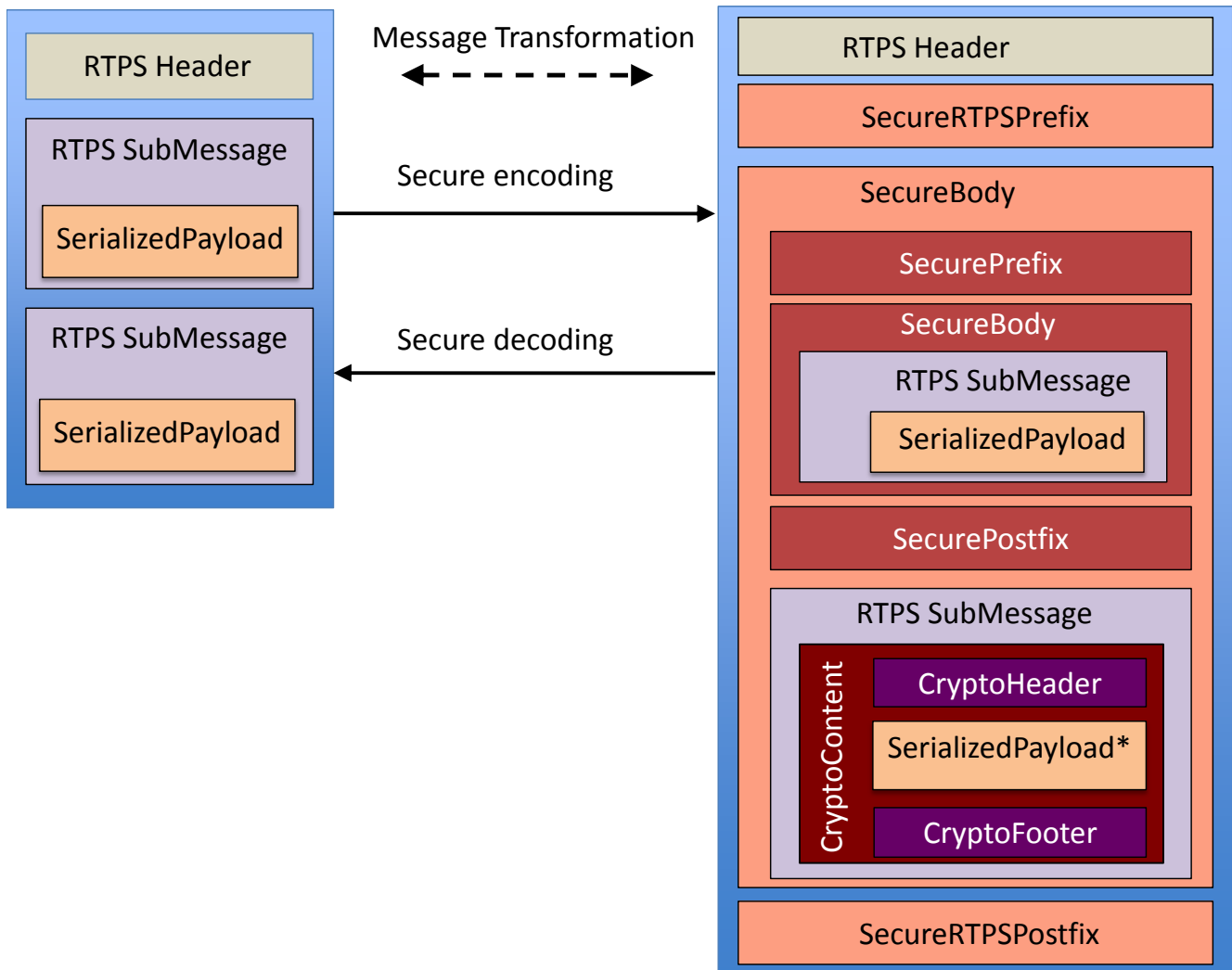
Figure 6 **– RTPS message transformations**

7.3.6.5   RTPS Submessage: SecurePostfixSubMsg

This specification introduces the RTPS submessage: `SecurePostfixSubMsg`. The format of the `SecurePostfixSubMsg` complies with the RTPS `SubMessage` format mandated in the RTPS specification. As such it consists of the RTPS `SubmessageHeader` followed by a set of RTPS `SubmessageElement` elements.

7.3.6.5.1   Purpose

The `SecurePostfixSubMsg` submessage is used to authenticate the RTPS Submessage that preceeds it.

7.3.6.5.2   Content

The elements that form the structure of the RTPS `SecurePostfixSubMsg` are described in the table below.

Table 6 – SecurePostfixSubMsg class

Element	Type	Meaning
SEC_POSTFIX	SubmessageKind	The presence of this field is common to RTPS submessages. It identifies the kind of submessage. The value indicates it is a SecurePostfixSubMsg.
submessageLength	ushort	The presence of this field is common to RTPS submessages. It identifies the length of the submessage.
EndianessFlag	SubmessageFlag	Appears in the Submessage header flags. Indicates endianess.
crypto_footer	CryptoFooter	Provides information on the results of the transformation performed, typically a list of authentication tags. The contents are specific to the Plugin Implementation and the value of the transformation_id contained on the related SecurePrefixSubMsg.

7.3.6.5.3 Validity

The RTPS `Submessage` is invalid if the ***submessageLength*** in the Submessage header is too small.

The RTPS `Submessage` is invalid if there is no `SecurePrefixSubMsg`. Immediately before the RTPS submessage that preceeds the `SecurePostfixSubMsg`. This `SecurePrefixSubMsg` is referred to as the *related* the `SecurePrefixSubMsg`.

7.3.6.5.4 Logical Interpretation

The `SecurePostfixSubMsg` provides a way to authenticate the validity and origin of the RTPS SubMessage that preceeds the `SecurePrefixSubMsg`. The Cryptographic transformation applied is identified in the *related* `SecurePrefixSubMsg`.

7.3.6.6   RTPS Submessage: SecureRTPSPrefixSubMsg

This specification introduces the RTPS submessage: `SecureRTPSPrefixSubMsg`. The format of the `SecurePrefixSubMsg` complies with the RTPS `SubMessage` format mandated in the RTPS specification. It consists of the RTPS `SubmessageHeader` followed by a set of RTPS `SubmessageElement` elements.

7.3.6.6.1   Purpose

The `SecureRTPSPrefixSubMsg` submessage is used as prefix to wrap a complete RTPS message in such a way that its contents are secured via encryption, message authentication, and/or digital signatures.

7.3.6.6.2   Content

The elements that form the structure of the RTPS `SecureRTPSPrefixSubMsg` are described in the table below.

Table 7 – SecureRTPSPrefixSubMsg class

Element	Type	Meaning
SRTPS_PREFIX	SubmessageKind	The presence of this field is common to RTPS submessages. It identifies the kind of submessage.

The value indicates it is a SecureRTPSPrefixSubMsg. |
submessageLength	ushort	The presence of this field is common to RTPS submessages. It identifies the length of the submessage.
EndianessFlag	SubmessageFlag	Appears in the Submessage header flags. Indicates endianess.
transformation_id	CryptoTransformIdentifier	Identfies the kind of transformation performed on the RTPS Subumessages that follow up to the SRTPS_POSTFIX submessage.
plugin_crypto_header_extra	octet[]	Provides further information on the transformation performed. The contents are specific to the Plugin Implementation and the value of the transformation_id.

7.3.6.6.3   Validity

The RTPS `Submessage` is invalid if the **submessageLength** in the Submessage header is too small.

The `SecureRTPSPrefixSubMsg` shall immediately follow the RTPS Header.

7.3.6.6.4   Logical Interpretation

The `SecureRTPSPrefixSubMsg` provides a way to prefix a list of RTPS Submessages so that they can be secured.

A `SecureRTPSPrefixSubMsg` shall be followed by a list of RTPS `Submessages` which themselves shall be followed by a `SecureRTPSPostfixSubMsg`.

7.3.6.7   RTPS Submessage: SecureRTPSPostfixSubMsg

This specification introduces the RTPS submessage: `SecureRTPSPostfixSubMsg`. The format of the `SecureRTPSPostfixSubMsg` complies with the RTPS `SubMessage` format mandated in the RTPS specification. As such it consists of the RTPS `SubmessageHeader` followed by a set of RTPS `SubmessageElement` elements.

7.3.6.7.1 Purpose

The `SecureRTPSPostfixSubMsg` submessage is used to authenticate the RTPS Submessages that appear between the preceeding `SecureRTPSPostfixSubMsg` and the `SecureRTPSPostfixSubMsg`.

7.3.6.7.2 Content

The elements that form the structure of the `SecureRTPSPostfixSubMsg` are described in the table below.

Table 8 – SecurePostfixSubMsg class

Element	Type	Meaning
SRTPS_POSTFIX	SubmessageKind	The presence of this field is common to RTPS submessages. It identifies the kind of submessage. The value indicates it is a SecureRTPSPostfixSubMsg.
submessageLength	ushort	The presence of this field is common to RTPS submessages. It identifies the length of the submessage.
EndianessFlag	SubmessageFlag	Appears in the Submessage header flags. Indicates endianess.
crypto_footer	CryptoFooter	Provides information on the results of the transformation performed, typically a list of authentication tags. The contents are specific to the Plugin Implementation and the value of the transformation_id contained on the related SecureRTPSPrefixSubMsg.

7.3.6.7.3 Validity

The RTPS `Submessage` is invalid if the **submessageLength** in the Submessage header is too small.

The RTPS `SecureRTPSPostfixSubMsg` is invalid if there is no `SecureRTPSPrefixSubMsg` following the RTPS Header. This `SecureRTPSPrefixSubMsg` is referred to as the *related* `SecureRTPSPrefixSubMsg`.

7.3.6.7.4 Logical Interpretation

The `SecureRTPSPostfixSubMsg` provides a way to authenticate the validity and origin of the list of RTPS Submessages between the related `SecureRTPSPrefixSubMsg` and the `SecureRTPSPrefixSubMsg`. The Cryptographic transformation applied is identified in the *related* `SecureRTPSPrefixSubMsg`.

7.3.7 Mapping to UDP/IP PSM

The DDS-RTPS specification defines the RTPS protocol in terms of a platform-independent model (PIM) and then maps it to a UDP/IP transport PSM (see clause 9, "Platform Specific Model (PSM): UDP/IP" of the DDS-RTPS specification [2]).

Sub clause 7.3.7 does the same thing for the new RTPS submessage elements and submessages introduced by the DDS Security specification.

7.3.7.1 Mapping of the EntityIds for the Builtin DataWriters and DataReaders

Sub clause 7.4 defines a set of builtin Topics and corresponding DataWriter and DataReader entities that shall be present on all compliant implementations of the DDS Security specification. The corresponding EntityIds used when these endpoints are used on the UDP/IP PSM are given in the table below.

Table 9 – EntityId values for secure builtin data writers and data readers

Entity	EntityId_t name	EntityId_t value
SEDPbuiltinPublicationsSecure Writer	ENTITYID_SEDP_BUILTIN_PUBLICATIO NS_SECURE_WRITER	{{ff, 00, 03}, c2}
SEDPbuiltinPublicationsSecure Reader	ENTITYID_SEDP_BUILTIN_PUBLICATIO NS_SECURE_READER	{{ff, 00, 03}, c7}
SEDPbuiltinSubscriptionsSecur eWriter	ENTITYID_SEDP_BUILTIN_SUBSCRIPTI ONS_SECURE_WRITER	{{ff, 00, 04}, c2}
SEDPbuiltinSubscriptionsSecur eReader	ENTITYID_ SEDP_BUILTIN_ SUBSCRIPTIONS_SECURE_READER	{{ff, 00, 04}, c7}
BuiltinParticipantMessageSecu reWriter	ENTITYID_P2P_BUILTIN_PARTICIPANT_ MESSAGE_SECURE_WRITER	{{ff, 02, 00}, c2}
BuiltinParticipantMessageSecu reReader	ENTITYID_P2P_BUILTIN_PARTICIPANT_ MESSAGE_SECURE_READER	{{ff, 02, 00}, c7}
BuiltinParticipantStatelessMes sageWriter	ENTITYID_P2P_BUILTIN_PARTICIPANT_ STATELESS_WRITER	{{00, 02, 01}, c3}
BuiltinParticipantStatelessMes sageReader	ENTITYID_P2P_BUILTIN_PARTICIPANT_ STATELESS_READER	{{00, 02, 01}, c4}
BuiltinParticipantVolatileMess ageSecureWriter	ENTITYID_P2P_BUILTIN_PARTICIPANT_ VOLATILE_SECURE_WRITER	{{ff, 02, 02}, c3}
BuiltinParticipantVolatileMess ageSecureReader	ENTITYID_P2P_BUILTIN_PARTICIPANT_ VOLATILE_SECURE_READER	{{ff, 02, 02}, c4}
SPDPbuiltinParticipantsSecure Writer	ENTITYID_SPDP_RELIABLE_BUILTIN_P ARTICIPANT_SECURE_WRITER	{{ff, 01, 01}, c2}
SPDPbuiltinParticipantsSecure Reader	ENTITYID_SPDP_RELIABLE _BUILTIN_PARTICIPANT_SECURE_READ ER	{{ff, 01, 01}, c7}

7.3.7.2   Mapping of the CryptoTransformIdentifier Type

The UDP/IP PSM maps the `CryptoTransformIdentifier` to the following extended IDL structure:

```
@extensibility(FINAL)
struct CryptoTransformIdentifier {
    octet transformation_kind[4];
    octet transformation_key_id[4];
};
```

7.3.7.3 Mapping of the CryptoHeader SubmessageElement

A `CryptoHeader SubmessageElement` contains the information that identifies a cryptographic transformation. The `CryptoHeader` shall start with the `CryptoTransformIdentifier` and be followed by a plugin-specific ***plugin_crypto_header_extra*** returned by the encoding transformation.

The UDP/IP PSM maps the `CryptoHeader` to the following extended IDL structure:

```
@extensibility(APPENDABLE)
struct CryptoHeader : CryptoTransformIdentifier  {
    // Extra plugin-specific information added below
    // CryptoHeader   plugin_crypto_header_extra;
};
```

The UDP/IP wire representation for the `CryptoHeader` shall be:

```
0...2...........8...............16.............24...............32
+--------------+--------------+--------------+--------------+
|                  octet  transformation_kind[4]            |
+--------------+--------------+--------------+--------------+
+                  octet  transformation_key_id[4]          +
+--------------+--------------+--------------+--------------+
|                                                           |
~                  octet  plugin_crypto_header_extra[]      ~
|                                                           |
+--------------+--------------+--------------+--------------+
```

7.3.7.4 Mapping of the CryptoFooter SubmessageElement

A `CryptoFooter SubmessageElement` contains the information that authenticates the result of a cryptographic transformation. The `CryptoFooter` contains a plugin-specific ***plugin_crypto_footer*** returned by the encoding transformation.

The UDP/IP wire representation for the `CryptoFooter` shall be:

```
0...2...........8...............16.............24...............32
+--------------+--------------+--------------+--------------+
|                                                           |
~                  octet  plugin_crypto_footer[]            ~
|                                                           |
+--------------+--------------+--------------+--------------+
```

7.3.7.5 SecureBodySubMsg Submessage

7.3.7.5.1 Wire Representation

The UDP/IP wire representation for the `SecureBodySubMsg` shall be:

```
0...2...........8...............16..............24...............32
+--------------+--------------+--------------+--------------+
| SEC_BODY     |X|X|X|X|X|X|X|E|    octetsToNextHeader       |
+--------------+--------------+--------------+--------------+
|                                                          |
+             CryptoContent  crypto_content                +
|                                                          |
+--------------+--------------+--------------+--------------+
```
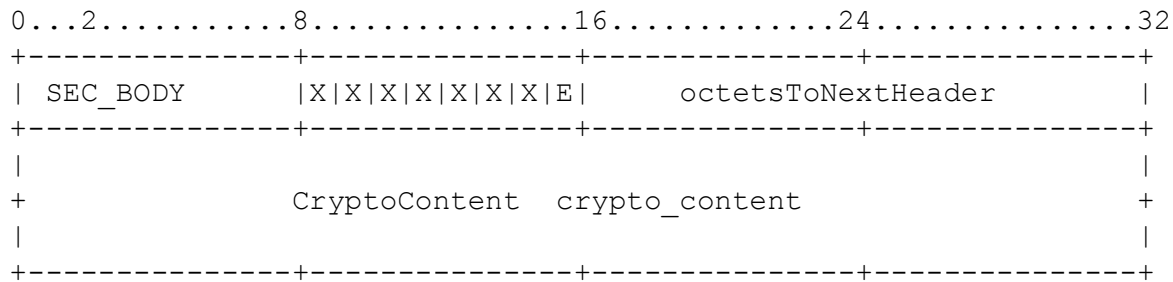
7.3.7.5.2 Submessage Id

The `SecureBodySubMsg` shall have the *submessageId* set to the value 0x30.

7.3.7.5.3 Flags in the Submessage Header

The `SecureBodySubMsg` only uses the EndiannessFlag.

7.3.7.6 SecurePrefixSubMsg Submessage

7.3.7.6.1 Wire Representation

The UDP/IP wire representation for the `SecurePrefixSubMsg` shall be:

```
0...2...........8...............16..............24...............32
+--------------+--------------+--------------+--------------+
| SEC_PREFIX   |X|X|X|X|X|X|X|E|    octetsToNextHeader       |
+--------------+--------------+--------------+--------------+
|                                                          |
+             CryptoHeader  crypto_header                   +
|                                                          |
+--------------+--------------+--------------+--------------+
```
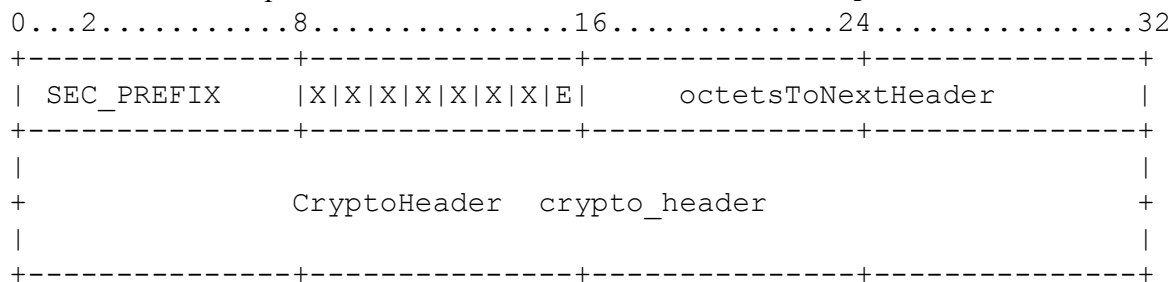
7.3.7.6.2 Submessage Id

The `SecurePrefixSubMsg` shall have the *submessageId* set to the value 0x31 and referred by the symbolic name `SEC_PREFIX`.

7.3.7.6.3 Flags in the Submessage Header

The `SecurePrefixSubMsg` only uses the EndiannessFlag.

7.3.7.7 SecurePostfixSubMsg Submessage

7.3.7.7.1 Wire Representation

The UDP/IP wire representation for the `SecurePostfixSubMsg` shall be:

```
0...2...........8...............16..............24...............32
+--------------+--------------+--------------+--------------+
| SEC_POSTFIX  |X|X|X|X|X|X|X|E|    octetsToNextHeader       |
+--------------+--------------+--------------+--------------+
|                                                           |
+              CryptoFooter  crypto_footer                  +
|                                                           |
+--------------+--------------+--------------+--------------+
```
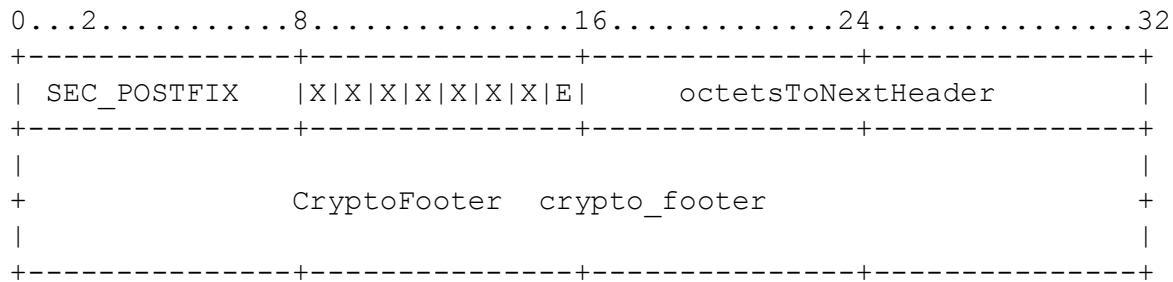
7.3.7.7.2 Submessage Id

The `SecurePostfixSubMsg` shall have the *submessageId* set to the value 0x32 and referred by the symbolic name `SEC_POSTFIX`.

7.3.7.7.3 Flags in the Submessage Header

The `SecurePostfixSubMsg` only uses the EndiannessFlag.

7.3.7.8 SecureRTPSPrefixSubMsg Submessage

7.3.7.8.1 Wire Representation

The UDP/IP wire representation for the `SecureRTPSPrefixSubMsg` shall be:

```
0...2...........8...............16..............24...............32
+--------------+--------------+--------------+--------------+
| SRTPS_PREFIX |X|X|X|X|X|X|X|E|    octetsToNextHeader       |
+--------------+--------------+--------------+--------------+
|                                                           |
+              CryptoHeader  crypto_header                  +
|                                                           |
+--------------+--------------+--------------+--------------+
```
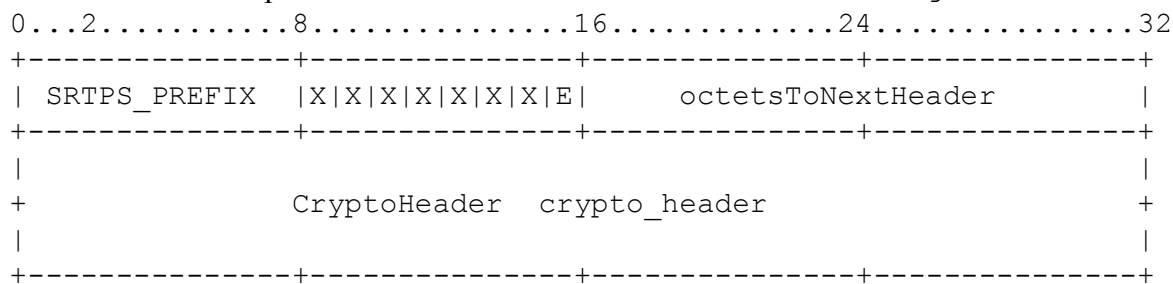
7.3.7.8.2 Submessage Id

The `SecureRTPSPrefixSubMsg` shall have the *submessageId* set to the value 0x33 and referred by the symbolic name `SRTPS_PREFIX`.

7.3.7.8.3 Flags in the Submessage Header

The `SecureRTPSPrefixSubMsg` only uses the EndiannessFlag.

7.3.7.9 SecureRTPSPostfixSubMsg Submessage

7.3.7.9.1 Wire Representation

The UDP/IP wire representation for the `SecureRTPSPostfixSubMsg` shall be:
```
0...2..........8................16..............24...............32
+--------------+--------------+--------------+--------------+
| SRTPS_POSTFIX |X|X|X|X|X|X|E|      octetsToNextHeader      |
+--------------+--------------+--------------+--------------+
|                                                          |
+          CryptoFooter  crypto_footer                     +
|                                                          |
+--------------+--------------+--------------+--------------+
```
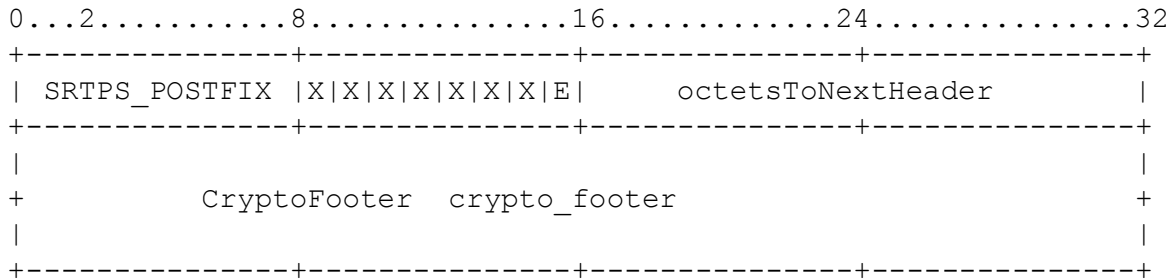
7.3.7.9.2 Submessage Id

The `SecureRTPSPostfixSubMsg` shall have the **submessageId** set to the value 0x34 and referred by the symbolic name `SRTPS_POSTFIX`.

7.3.7.9.3 Flags in the Submessage Header

The `SecureRTPSPostfixSubMsg` only uses the EndiannessFlag.

7.4 DDS Support for Security Plugin Information Exchange

In order to perform their function, the security plugins associated with different DDS `DomainParticipant` entities need to exchange information representing things such as Identity and Permissions of the `DomainParticipant` entities, authentication challenge messages, tokens representing key material, etc.

DDS already has several mechanisms for information exchange between `DomainParticipant` entities. Notably the builtin `DataWriter` and `DataReader` entities used by the Simple Discovery Protocol (see sub clause 8.5 of the DDS Interoperability Wire Protocol [2]) and the **BuiltinParticipantMessageWriter** and **BuiltinParticipantMessageReader** (see sub clause 9.6.2.1 of the DDS Interoperability Wire Protocol [2]).

Where possible, this specification tries to reuse and extend existing DDS concepts and facilities so that they can fulfill the needs of the security plugins, rather than defining entirely new ones. This way, the Security Plugin implementation can be simplified and it does not have to implement a separate messaging protocol.

7.4.1 Secure builtin Discovery Topics

7.4.1.1 Background (Non-Normative)

DDS discovery information is sent using builtin DDS `DataReaders` and `DataWriters`. These are regular DDS `DataReaders` and `DataWriters`, except they are always present in the system and their `Topic` names, associated data types, QoS, and RTPS `EntityIds` are all specified as part of the DDS and RTPS specifications, so they do not need to be discovered.

The DDS specification defines three discovery builtin `Topic` entities: the **DCPSParticipants** used to discover the presence of `DomainParticipants`, the **DCPSPublications** used to discover

`DataWriters`, and the **DCPSSubscriptions** used to discover `DataReaders` (see sub clause 8.5 of the DDS Interoperability Wire Protocol [2]).

Much of the discovery information could be considered sensitive in secure DDS systems. Knowledge of things like the `Topic` names that an application is publishing or subscribing to could reveal sensitive information about the nature of the application. In addition, the integrity of the discovery information needs to be protected against tampering, since it could cause erroneous behaviors or malfunctions.

One possible approach to protecting discovery information would be to require that the discovery builtin `Topic` entities always be protected via encryption and message authentication. However, this would entail the problems explained below.

The **DCPSParticipants** builtin Topic is used to bootstrap the system, detect the presence of `DomainParticipant` entities, and kick off subsequent information exchanges and handshakes. It contains the bare minimum information needed to establish protocol communications (addresses, port numbers, version number, vendor IDs, etc.). If this `Topic` were protected, the Secure DDS system would have to create an alternative mechanism to bootstrap detection of other participants and gather the same information—which needs to happen prior to being able to perform mutual authentication and exchange of key material. This mechanism would, in essence, duplicate the information in the **DCPSParticipants** builtin `Topic`. Therefore, it makes little sense to protect the **DCPSParticipants** builtin `Topic`. A better approach is to augment the information sent using the **DCPSParticipants** builtin `Topic` with any additional data the Secure DDS system needs for bootstrapping communications (see 7.4.1.3).

Secure DDS systems need to co-exist in the same network and, in some cases, interoperate with non-secure DDS systems. There may be systems built using implementations compliant with the DDS Security specification, which do not need to protect their information. Or there may be systems implemented with legacy DDS implementations that do not support DDS Security. In this situation, the fact that a secure DDS implementation is present on the network should not impact the otherwise correct behavior of the non-secure DDS systems. In addition, even in secure systems not all Topics are necessarily sensitive, so it is desirable to provide ways to configure a DDS Secure system to have Topics that are "unprotected" and be able to communicate with non-secure DDS systems on those "unprotected" Topics.

To allow co-existence and interoperability between secure DDS systems and DDS systems that do not implement DDS security, secure DDS systems must retain the same builtin Topics as the regular DDS systems (with the same GUIDs, topics names, QoS, and behavior). Therefore, to protect the discovery and liveliness information of Topics that are considered sensitive, Secure DDS needs to use additional builtin discovery Topics protected by the DDS security mechanisms.

7.4.1.2 Extending the Data Types used by DDS Discovery

The DDS Interoperability Wire Protocol specifies the serialization of the data types used for the discovery of builtin Topics (**ParticipantBuiltinTopicData**, **PublicationBuiltinTopicData**, and **SubscriptionBuiltinTopicData)** using a representation called a *ParameterList*. Although this description precedes the DDS-XTYPES specification, the serialization format matches the Extended CDR representation defined in DDS-XTYPES for data types declared with MUTABLE extensibility. This allows the data type associated with discovery topics to be extended without breaking interoperability.

Given that DDS-XTYPES formalized the *ParameterList* serialization approach, first defined in the DDS Interoperability and renamed it to "Extended CDR," this specification will use the DDS Extensible Types notation to define the data types associated with the builtin Topics. This does not imply that compliance to the DDS-XTYPES is required to comply with DDS Security. All that is required is to serialize the specific data types defined here according to the format described in the DDS-XTYPES specification.

7.4.1.3 Reserved RTPS parameter IDs

This specification reserves the RTPS Simple Discovery Protocol ParameterIDs in the range: 0x1000 to 0x1FFF and 0x5000 to 0x5FFF.

The second interval covers the same range of parametersID, except they have the must-understand bit set.

This reserved range applies to RTPS version 2.3 (see 7.3.6.1) and higher minor revisions of RTPS. Future revisions of the DDS-RTPS specification shall take this fact into consideration.

7.4.1.4 Extension to RTPS Standard DCPSParticipants Builtin Topic

The DDS specification specifies the existence of the **DCPSParticipants** builtin `Topic` and a corresponding builtin `DataWriter` and `DataReader` to communicate this `Topic`. These endpoints are used to discover `DomainParticipant` entities.

The data type associated with the **DCPSParticipants** builtin `Topic` is **ParticipantBuiltinTopicData**, defined in sub clause 7.1.5 of the DDS specification [1].

The DDS Interoperability Wire Protocol specifies the serialization of **ParticipantBuiltinTopicData.** The format used is what the DDS Interoperability Wire Protocol calls a *ParameterList* whereby each member of the **ParticipantBuiltinTopicData** is serialized using CDR but preceded in the stream by the serialization of a short ParameterID identifying the member, followed by another short containing the length of the serialized member, followed by the serialized member. See sub clause 8.3.5.9 of the DDS Interoperability Wire Protocol [2]. This serialization format allows the **ParticipantBuiltinTopicData** to be extended without breaking interoperability.

This DDS Security specification adds several new members to the **ParticipantBuiltinTopicData** structure. The member types and the **ParameterIDs** used for the serialization are described below.

Table 10 – Additional parameter IDs in ParticipantBuiltinTopicData

Member name	*Member type*	*Parameter ID name*	*Parameter ID value*
identity_token	IdentityToken (see 7.2.4)	PID_IDENTITY_TOKEN	0x1001
permissions_token	PermissionsToken (see 7.2.4)	PID_PERMISSIONS_TOKEN	0x1002
property	PropertyQosPolicy	PID_PROPERTY_LIST (See Table 9.12 of DDS-RTPS)	0x0059 (See Table 9.12 of DDS-RTPS)
security_info	ParticipantSecurityInfo (see 7.2.7)	PID_PARTICIPANT_SECURITY_INFO	0x1005

```
@extensibility(MUTABLE)
struct ParticipantBuiltinTopicData: DDS::ParticipantBuiltinTopicData {
    @id(0x1001) IdentityToken       identity_token;
    @id(0x1002) PermissionsToken   permissions_token;
    @id(0x1005) ParticipantSecurityInfo  security_info;
};
```

Only the `Property_t` and `BinaryProperty_t` elements having the propagate member set to TRUE are serialized. Furthermore as indicated by the `@non-serialized` annotation the serialization of the `Property_t` and `BinaryProperty_t` elements shall omit the serialization of the *propagate* member. That is they are serialized as if the type definition did not contain the *propagate* member. This is consistent with the data-type definition for Property_t specific in the DDS-RTPS specification (see Table 9.12 of DDS-RTPS). Even if it is not present in the serialized data, the receiver will set the *propagate* member to TRUE.

Note that according to DDS-RTPS the PID_PROPERTY_LIST is associated with a single `PropertySeq` rather than the `PropertyQosPolicy`, which is a structure that contains two sequences. This does not cause any interoperability problems because the containing `ParticipantBuiltinTopicData` has mutable extensibility.

The DDS Interoperability Wire Protocol specifies that the *ParticipantBuiltinTopicData* shall contain the attribute called *availableBuiltinEndpoints* that is used to announce the builtin endpoints that are available in the `DomainParticipant`. See clause 8.5.3.2 of the DDS Interoperability Wire Protocol [2]. The type for this attribute is an array of *BuiltinEndpointSet_t*. For the UDP/IP PSM the *BuiltinEndpointSet_t* is mapped to a bitmap represented as type **long**. Each builtin endpoint is represented as a bit in this bitmap with the bit values defined in Table 9.4 (clause 9.3.2) of the DDS Interoperability Wire Protocol [2].

This DDS Security specification reserves additional bits to indicate the presence of the corresponding built-in end points listed in clause 7.4.5. These bits shall be set on the *availableBuiltinEndpoints*. The bit that encodes the presence of each individual endpoint is defined in Table 11 below.

Table 11 – Mapping of the additional builtin endpoints added by DDS security to the availableBuiltinEndpoints

Builtin Endpoint	Bit in the ParticipantBuiltinTopicData availableBuiltinEndpoints
SEDPbuiltinPublicationsSecureWriter *SEDPbuiltinPublicationsSecureReader* See clause 7.4.1.7	(0x00000001 << 16) (0x00000001 << 17)
SEDPbuiltinSubscriptionsSecureWriter *SEDPbuiltinSubscriptionsSecureReader* See clause 7.4.1.8	(0x00000001 << 18) (0x00000001 << 19)
BuiltinParticipantMessageSecureWriter *BuiltinParticipantMessageSecureReader* See clause 7.4.2	(0x00000001 << 20) (0x00000001 << 21)
BuiltinParticipantStatelessMessageWriter *BuiltinParticipantStatelessMessageReader* See clause 7.4.3	(0x00000001 << 22) (0x00000001 << 23)
BuiltinParticipantVolatileMessageSecureWriter *BuiltinParticipantVolatileMessageSecureReader* See clause 7.4.4	(0x00000001 << 24) (0x00000001 << 25)
SPDPbuiltinParticipantSecureWriter *SPDPbuiltinParticipantSecureReader* See clause 7.4.1.6	(0x00000001 << 26) (0x00000001 << 27)

7.4.1.5   Extension to RTPS Standard DCPSPublications and DCPSSubscriptions Builtin Topics

The DDS specification specifies the existence of the *DCPSPublications* and *DCPSSubscriptions* builtin `Topics` and a corresponding builtin `DataWriters` and `DataReaders` to communicate these `Topics`. These endpoints are used to discover `DataWriter` and `DataReader` entities.

The data type associated with the *DCPSPublications* and *DCPSSubscriptions* builtin `Topic` are *PublicationBuiltinTopicData* and *SubscriptionBuiltinTopicData*, defined in sub clause 7.1.5 of the DDS specification.

The DDS Interoperability Wire Protocol specifies the serialization of *PublicationBuiltinTopicData* and *SubscriptionBuiltinTopicData*.

The format used is what the DDS Interoperability Wire Protocol calls a `ParameterList` whereby each member of the *PublicationBuiltinTopicData* and *SubscriptionBuiltinTopicData* is serialized using CDR but preceded in the stream by the serialization of a short `ParameterID` identifying the member, followed by another short containing the length of the serialized member, followed by the serialized member. See sub clause

8.3.5.9 of the DDS Interoperability Wire Protocol [2]. This serialization format allows the ***PublicationBuiltinTopicData*** and ***SubscriptionBuiltinTopicData*** to be extended without breaking interoperability.

This DDS Security specification adds a new member to the ***PublicationBuiltinTopicData*** and ***SubscriptionBuiltinTopicData*** structure. The member types and the `ParameterIDs` used for the serialization are described below.

Table 12 – Additional parameter IDs in PublicationBuiltinTopicData

Member name	*Member type*	*Parameter ID name*	*Parameter ID value*
security_info	EndpointSecurityInfo (See 7.2.8)	PID_ENDPOINT_SECURITY_INFO	0x1004

```
@extensibility(MUTABLE)
struct PublicationBuiltinTopicData: DDS::PublicationBuiltinTopicData {
    @id(0x1004) EndpointSecurityInfo  security_info;
};

@extensibility(MUTABLE)
struct SubscriptionBuiltinTopicData: DDS::SubscriptionBuiltinTopicData {
    @id(0x1004) EndpointSecurityInfo  security_info;
};
```

7.4.1.6   New DCPSParticipantSecure Builtin Topic

As described in clause 7.4.1.4, the ***DCPSParticipants*** builtin `Topic` and a corresponding builtin `DataWriter` and `DataReader` are used to discover `DomainParticipant` entities.

Implementations of the DDS Security shall use that same ***DCPSParticipants*** builtin `Topic` to announce the `DomainParticipant` information. This is used for bootstrapping authentication and allowing discovery of non-secure applications.

Implementations of the DDS Security specification shall have an additional builtin `Topic` referred to as ***DCPSParticipantsSecure*** and associated builtin `DataReader` and `DataWriter` entities to communicate the `DomainParticipant` information securely.

The `Topic` name for the ***DCPSParticipantsSecure*** `Topic` shall be "DCPSParticipantsSecure".

The data type associated with the ***DCPSParticipantsSecure*** `Topic` shall be ***ParticipantBuiltinTopicDataSecure***, defined to be the same as the ***ParticipantBuiltinTopicData*** defined in clause 7.4.1.4, except the structure has the additional optional member ***identity_status_token*** with the ***ParameterId*** described below.

Table 13 – Additional parameter IDs in ParticipantBuiltinTopicDataSecure

Member name	Member type	Parameter ID name	Parameter ID value
identity_status_token	IdentityStatusToken	PID_IDENTITY_STATUS_TOKEN	0x1006

```
@extensibility(MUTABLE)
struct ParticipantBuiltinTopicDataSecure: ParticipantBuiltinTopicData {
    @id(0x1006) @optional IdentityStatusToken identity_status_token;
};
```

The QoS associated with the *DCPSParticipantsSecure* builtin `Topic` shall be the same as for the *DCPSPublications* and *DCPSSubscriptions* builtin `Topic`. Note that is not the same as the *DCPSParticipants* `Topic`. Among other differences, the *DCPSParticipantsSecure* has `ReliabilityQosPolicy` *kind* set to RELIABLE.

The builtin `DataWriter` for the *DCPSParticipantsSecure* `Topic` shall be referred to as the *SPDPbuiltinParticipantsSecureWriter*. The builtin `DataReader` for the *DCPSParticipantsSecure* `Topic` shall be referred to as the *SPDPbuiltinParticipantsSecureReader.*

The RTPS `EntityId_t` associated with the *SPDPbuiltinParticipantsSecureWriter* and *SPDPbuiltinParticipantsSecureReader* shall be as specified in 7.4.5.

The *ParticipantBuiltinTopicData* contains information, such as participant Locators, which may change at run-time. These changes shall be sent using the *DCPSParticipantsSecure* builtin `Topic`. The deletion of a DomainParticipant shall also be sent using the *DCPSParticipantsSecure* builtin `Topic`.

After authentication has completed successfully a `DomainParticipant` shall ignore any changes to the *ParticipantBuiltinTopicData* (including dispose messages) received on the *DCPSParticipants* builtin `Topic` from the authenticated `DomainParticipant`. It may, however, rely on these messages to maintain the liveliness of the remote `DomainParticipant`. It should only process *ParticipantBuiltinTopicData* messages containing data changes or status changes (dispose or unregister) if they are received over the *DCPSParticipantsSecure* builtin `Topic`.

7.4.1.7   New DCPSPublicationsSecure Builtin Topic

The DDS specification specifies the existence of the *DCPSPublications* builtin `Topic` with topic name "DCPSPublications" and corresponding builtin `DataWriter` and `DataReader` entities to communicate on this `Topic`. These endpoints are used to discover non-builtin `DataWriter` entities.

The data type associated with the *DCPSPublications* `Topic` is *PublicationBuiltinTopicData*, defined in sub clause 7.1.5 of the DDS specification.

Implementations of the DDS Security shall use that same *DCPSPublications* `Topic` to communicate the `DataWriter` information for `Topic` entities that **are not** considered sensitive.

Implementations of the DDS Security specification shall have an additional builtin `Topic` referred to as *DCPSPublicationsSecure* and associated builtin `DataReader` and `DataWriter` entities to communicate the `DataWriter` information for `Topic` entities that **are** considered sensitive.

The determination of which `Topic` entities are considered sensitive shall be specified by the `AccessControl` plugin.

The `Topic` name for the ***DCPSPublicationsSecure*** `Topic` shall be "DCPSPublicationsSecure".

The data type associated with the ***DCPSPublicationsSecure*** `Topic` shall be ***PublicationBuiltinTopicDataSecure***, defined to be the same as the ***PublicationBuiltinTopicData*** structure used by the ***DCPSPublications*** `Topic,` except the structure has the additional member ***data_tags*** with the ***ParameterId*** described below.

Table 14 – Additional parameter IDs in PublicationBuiltinTopicDataSecure

Member name	*Member type*	*Parameter ID name*	*Parameter ID value*
data_tags	DataTags	PID_DATA_TAGS	0x1003

```
@extensibility(MUTABLE)
struct PublicationBuiltinTopicDataSecure: PublicationBuiltinTopicData {
    @id(0x1003) DataTags data_tags;
};
```

The QoS associated with the ***DCPSPublicationsSecure*** `Topic` shall be the same as for the ***DCPSPublications*** `Topic`.

The builtin `DataWriter` for the ***DCPSPublicationsSecure*** `Topic` shall be referred to as the ***SEDPbuiltinPublicationsSecureWriter***. The builtin `DataReader` for the ***DCPSPublicationsSecure*** `Topic` shall be referred to as the ***SEDPbuiltinPublicationsSecureReader***.

The RTPS `EntityId_t` associated with the ***SEDPbuiltinPublicationsSecureWriter*** and ***SEDPbuiltinPublicationsSecureReader*** shall be as specified in 7.4.5.

7.4.1.8   New DCPSSubscriptionsSecure Builtin Topic

The DDS specification specifies the existence of the ***DCPSSubscriptions*** builtin `Topic` with `Topic` name "DCPSSubscriptions" and corresponding builtin `DataWriter` and `DataReader` entities to communicate on this `Topic`. These endpoints are used to discover non-builtin `DataReader` entities.

The data type associated with the ***DCPSSubscriptions*** is ***SubscriptionBuiltinTopicData*** is defined in sub clause 7.1.5 of the DDS specification.

Implementations of the DDS Security specification shall use that same ***DCPSSubscriptions*** `Topic` to send the `DataReader` information for `Topic` entities that **are not** considered sensitive. The existence and configuration of `Topic` entities as non-sensitive shall be specified by the `AccessControl` plugin.

Implementations of the DDS Security specification shall have an additional builtin `Topic` referred to as ***DCPSSubscriptionsSecure*** and associated builtin `DataReader` and `DataWriter` entities to communicate the `DataReader` information for `Topic` entities that are considered sensitive.

The determination of which `Topic` entities are considered sensitive shall be specified by the `AccessControl` plugin.

The data type associated with the **DCPSSubscriptionsSecure** `Topic` shall be **SubscriptionBuiltinTopicDataSecure** defined to be the same as the **SubscriptionBuiltinTopicData** structure used by the **DCPSSubscriptions** `Topic`, except the structure has the additional member **data_tags** with the data type and **ParameterIds** described below.

Table 15 – Additional parameter IDs in SubscriptionBuiltinTopicDataSecure

Member name	Member type	Parameter ID name	Parameter ID value
data_tags	DataTags	PID_DATA_TAGS	0x1003

```
@extensibility(MUTABLE)
struct SubscriptionBuiltinTopicDataSecure: SubscriptionBuiltinTopicData {
    @id(0x1003) DataTags data_tags;
};
```

The QoS associated with the **DCPSSubscriptionsSecure** `Topic` shall be the same as for the **DCPSSubscriptions** `Topic`.

The builtin `DataWriter` for the **DCPSSubscriptionsSecure** `Topic` shall be referred to as the **SEDPbuiltinSubscriptionsSecureWriter**. The builtin `DataReader` for the **DCPSPublicationsSecure** `Topic` shall be referred to as the **SEDPbuiltinSubscriptionsSecureReader**.

The RTPS `EntityId_t` associated with the **SEDPbuiltinSubscriptionsSecureWriter** and **SEDPbuiltinSubscriptionsSecureReader** shall be as specified in 7.4.5.

7.4.2 New DCPSParticipantMessageSecure builtin Topic

The DDS Interoperability Wire Protocol specifies the **BuiltinParticipantMessageWriter** and **BuiltinParticipantMessageReader** (see sub clauses 8.4.13 and 9.6.2.1 of the DDS Interoperability Wire Protocol[2]). These entities are used to send information related to the LIVELINESS QoS. This information could be considered sensitive and therefore secure DDS systems need to provide an alternative protected way to send liveliness information.

The data type associated with these endpoints is **ParticipantMessageData** defined in sub clause 9.6.2.1 of the DDS Interoperability Wire Protocol specification [2].

To support coexistence and interoperability with non-secure DDS applications, implementations of the DDS Security specification shall use the same standard **BuiltinParticipantMessageWriter** and **BuiltinParticipantMessageReader** to communicate liveliness information on `Topic` entities that **are not** considered sensitive.

Implementations of the DDS Security specification shall have an additional **DCPSParticipantMessageSecure** builtin `Topic` and associated builtin `DataReader` and `DataWriter` entities to communicate the liveliness information for `Topic` entities that **are** considered sensitive.

The data type associated with the **DCPSParticipantMessageSecure** `Topic` shall be the same as the **ParticipantMessageData** structure.

The QoS associated with the **DCPSParticipantMessageSecure** `Topic` shall be the same as for the **DCPSParticipantMessage** `Topic` as defined in sub clause 8.4.13 of the DDS Interoperability Wire Protocol [2].

The builtin `DataWriter` for the ***DCPSParticipantMessageSecure*** `Topic` shall be referred to as the ***BuiltinParticipantMessageSecureWriter***. The builtin `DataReader` for the ***DCPSParticipantMessageSecure*** `Topic` shall be referred to as the ***BuiltinParticipantMessageSecureReader***.

The RTPS `EntityId_t` associated with the ***BuiltinParticipantMessageSecureWriter*** and ***BuiltinParticipantMessageSecureReader*** shall be as specified in 7.4.5.

According to clause 8.7.2.2.3 of DDSI-RTPS [2], if the `DataWriter LivelinessQos` policy is `MANUAL_BY_TOPIC_LIVELINESS_QOS`, liveliness is maintained sending data or heartbeats using the same RTPS `DataWriter`. The remaining settings for the `LivelinessQos` policy use the ***DCPSParticipantMessage*** `Topic` to maintain the `DataWriter` liveliness.

If a `DataWriter LivelinessQos` policy is `MANUAL_BY_TOPIC_LIVELINESS_QOS`, implementations compliant with DDS-Security shall use the same RTPS `DataWriter` for the liveliness heartbeats. The liveliness heartbeats shall be protected using the same means as the regular `DataWriter` heartbeats. That is, according to the setting of the `EndpointSecurityAttributes` ***is_submessage_protected*** attribute.

If the `DataWriter LivelinessQos` policy is `AUTOMATIC_LIVELINESS_QOS` or `MANUAL_BY_PARTICIPANT_LIVELINESS_QOS`, implementations compliant with DDS-Security shall send the liveliness heartbeats using either the ***DCPSParticipantMessage*** `Topic` or the ***DCPSParticipantMessageSecure*** `Topic`. The selection shall be done according to the setting of the `TopicSecurityAttributes` ***is_liveliness_protected***: It shall use the ***DCPSParticipantMessage*** `Topic` if ***is_liveliness_protected*** is set to **false**, otherwise it shall use the ***DCPSParticipantMessageSecure*** `Topic`.

7.4.3 New DCPSParticipantStatelessMessage builtin Topic

To perform mutual authentication between DDS `DomainParticipant` entities, the security plugins associated with those participants need to be able to send directed messages to each other. As described in 7.4.3.1 below, the mechanisms provided by existing DDS builtin `Topic` entities are not adequate for this purpose. For this reason, this specification introduces a new ***DCPSParticipantStatelessMessage*** builtin `Topic` and corresponding builtin `DataReader` and `DataWriter` entities to read and write the `Topic`.

7.4.3.1 Background: Sequence Number Attacks (non normative)

DDS has a builtin mechanism for participant-to-participant messaging: the ***BuiltinParticipantMessageWriter*** and ***BuiltinParticipantMessageReader*** (see sub clause 9.6.2.1 of the DDS Interoperability Wire Protocol [2]). However this mechanism cannot be used for mutual authentication because it relies on the RTPS reliability protocol and suffers from the sequence-number prediction vulnerability present in unsecured reliable protocols:

- The RTPS reliable protocol allows a `DataWriter` to send to a `DataReader Heartbeat` messages that advance the *first available sequence number* associated with the `DataWriter`. A `DataReader` receiving a `Heartbeat` from a `DataWriter` will advance its *first available sequence number* for that `DataWriter` and ignore any future messages it receives with sequence numbers lower than the *first available sequence number* for the `DataWriter`. The reliable `DataReader` will also ignore duplicate messages for that same sequence number.

- The behavior of the reliability protocol would allow a malicious application to prevent other applications from communicating by sending `Heartbeats` pretending to be from other `DomainParticipants` that contain large values of the *first available sequence number*. All the malicious application needs to do is learn the GUIDs of other applications, which can be done from observing the initial discovery messages on the wire, and use that information to create fake `Heartbeats`.

Stated differently: prior to performing mutual authentication and key exchange, the applications cannot rely on the use of encryption and message access codes to protect the integrity of the messages. Therefore, during this time window, they are vulnerable to this kind of sequence-number attack. This attack is present in most reliable protocols. Stream-oriented protocols such as TCP are also vulnerable to sequence-number-prediction attacks but they make it more difficult by using a random initial sequence number on each new connection and discarding messages with sequence numbers outside the window. This is something that RTPS cannot do given the data-centric semantics of the protocol.

In order to avoid this vulnerability, the Security plugins must exchange messages using writers and readers sufficiently robust to sequence number prediction attacks. The RTPS protocol specifies endpoints that meet this requirement: the RTPS `StatelessWriter` and `StatelessReader` (see 8.4.7.2 and 8.4.10.2 of the DDS Interoperability Wire Protocol [2]) but there are no DDS builtin endpoints that provide access to this underlying RTPS functionality.

7.4.3.2 BuiltinParticipantStatelessMessageWriter and BuiltinParticipantStatelessMessageReader

The DDS Security specification defines two builtin Endpoints: the ***BuiltinParticipantStatelessMessageWriter*** and the ***BuiltinParticipantStatelessMessageReader***. These two endpoints shall be present in compliant implementations of this specification. These endpoints are used to write and read the builtin ***DCPSParticipantStatelessMessage*** `Topic`.

The ***BuiltinParticipantStatelessMessageWriter*** is an RTPS Best-Effort `StatelessWriter` (see sub clause 8.4.7.2 of the DDS Interoperability Wire Protocol [2]).

The ***BuiltinParticipantStatelessMessageReader*** is an RTPS Best-Effort `StatelessReader` (see sub clause 8.4.10.2 of the DDS Interoperability Wire Protocol [2]).

The data type associated with these endpoints is `ParticipantStatelessMessage` defined below (see also 7.2.5):

```
typedef ParticipantStatelessMessage ParticipantGenericMessage;
```

The RTPS `EntityId_t` associated with the ***BuiltinParticipantStatelessMessageWriter*** and ***BuiltinParticipantStatelessMessageReader*** shall be as specified in 7.4.5.

7.4.3.3 Contents of the ParticipantStatelessMessage

The `ParticipantStatelessMessage` is intended as a holder of information that is sent point-to-point from a `DomainParticipant` to another.

The ***message_identity*** uniquely identifies each individual `ParticipantStatelessMessage`:

- The ***source_guid*** field within the ***message_identity*** shall be set to match the `GUID_t` of the ***BuiltinParticipantStatelessMessageWriter*** that writes the message.

- The *sequence_number* field within the *message_identity* shall start with the value set to one and be incremented for each different message sent by the ***BuiltinParticipantStatelessMessageWriter***.

The *related_message_identity* uniquely identifies another `ParticipantStatelessMessage` that is related to the message being processed. It shall be set to either the tuple {***GUID_UNKNOWN, 0}*** if the message is not related to any other message, or else set to match the *message_identity* of the related `ParticipantStatelessMessage`.

The *destination_participant_guid* shall contain either the value ***GUID_UNKNOWN*** (see sub clause 9.3.1.5 of the DDS Interoperability Wire Protocol [2]) or else the `GUID_t` of the destination `DomainParticipant`.

The *destination_endpoint_guid* provides a mechanism to specify finer granularity on the intended recipient of a message beyond the granularity provided by the *destination_participant_guid*. It can contain either ***GUID_UNKNOWN*** or else the GUID of a specific endpoint within destination `DomainParticipant`. The targeted endpoint is the one whose Endpoint (`DataWriter` or `DataReader`) `GUID_t` matches the *destination_endpoint_guid*.

The contents *message_data* depend on the value of the *message_class_id* and are defined in this specification in the sub clause that introduces each one of the pre-defined values of the `GenericMessageClassId`. See 7.4.3.5 and 7.4.3.6.

7.4.3.4   Destination of the ParticipantStatelessMessage

If the *destination_participant_guid* member is not set to ***GUID_UNKNOWN***, the message written is intended only for the ***BuiltinParticipantStatelessMessageReader*** belonging to the `DomainParticipant` with a matching Participant Key.

This is equivalent to saying that the ***BuiltinParticipantStatelessMessageReader*** has an implied content filter with the logical expression:

"destination_participant_guid == GUID_UNKNOWN

|| destination_participant_guid == BuiltinParticipantStatelessMessageReader.participant.guid"

Implementations of the specification can use this content filter or some other mechanism as long as the resulting behavior is equivalent to having this content filter.

If the *destination_endpoint_guid* member is not set to ***GUID_UNKNOWN***, the message written targets the specific endpoint within the destination `DomainParticipant` with a matching Endpoint Key.

7.4.3.5   Reserved values of ParticipantStatelessMessage GenericMessageClassId

This specification, including future versions of this specification reserves *GenericMessageClassId* values that start with the prefix "`dds.sec.`" (without quotes) .

The specification defines and uses the following specific values for the *GenericMessageClassId*:

```
#define GMCLASSID_SECURITY_AUTH_REQUEST              \
          "dds.sec.auth_request"
#define GMCLASSID_SECURITY_AUTH_HANDSHAKE            \
          "dds.sec.auth"
```

Additional values of the **GenericMessageClassId** may be defined with each plugin implementation.

7.4.3.6   Format of data within ParticipantStatelessMessage

Each value for the `GenericMessageClassId` uses different schema to store data within the generic attributes in the **message_data**.

7.4.3.6.1   Data for message class GMCLASSID_SECURITY_AUTH_HANDSHAKE

If `GenericMessageClassId` is `GMCLASSID_SECURITY_AUTH_HANDSHAKE` the **message_data** attribute shall contain the `HandshakeMessageTokenSeq` containing one element. The specific contents of the `HandshakeMessageToken` element shall be defined by the Authentication Plugin.

The **destination_participant_guid** shall be set to the `GUID_t` of the destination `DomainParticipant`.

The **destination_endpoint_guid** shall be set to **GUID_UNKNOWN**. This indicates that there is no specific endpoint targeted by this message: It is intended for the whole `DomainParticipant`.

The **source_endpoint_guid** shall be set to **GUID_UNKNOWN**.

7.4.3.6.2   Data for message class GMCLASSID_SECURITY_AUTH_REQUEST

If `GenericMessageClassId` is `GMCLASSID_SECURITY_AUTH_REQUEST` the **message_data** attribute shall contain a `AuthRequestMessageTokenSeq` containing one element. The specific contents of the `AuthRequestMessageToken` element shall be defined by the Authentication Plugin.

The **destination_participant_guid** shall be set to the `GUID_t` of the destination `DomainParticipant`.

The **destination_endpoint_guid** shall be set to **GUID_UNKNOWN**. This indicates that there is no specific endpoint targeted by this message: It is intended for the whole `DomainParticipant`.

The **source_endpoint_guid** shall be set to **GUID_UNKNOWN**.

7.4.4   New DCPSParticipantVolatileMessageSecure builtin Topic

7.4.4.1   Background (Non-Normative)

In order to perform key exchange between DDS `DomainParticipant` entities, the security plugins associated with those participants need to be able to send directed messages to each other using a reliable and secure channel. These messages are intended only for Participants that are currently in the system and therefore need a DURABILITY Qos of kind VOLATILE.

The existing mechanisms provided by DDS are not adequate for this purpose:

- The new **DCPSParticipantStatelessMessage** is not suitable because it is a stateless best-effort channel not protected by the security mechanisms in this specification and therefore requires the message data to be explicitly encrypted and signed prior to being given to the **ParticipantStatelessMessageWriter**.

- The new ***DCPSParticipantMessageSecure*** is not suitable because its QoS has DURABILITY kind TRANSIENT_LOCAL (see sub clause 8.4.13 of the DDS Interoperability Wire Protocol [2]) rather than the required DURABILITY kind VOLATILE.

For this reason, implementations of the DDS Security specification shall have an additional builtin `Topic` ***DCPSParticipantVolatileMessageSecure*** and corresponding builtin `DataReader` and `DataWriter` entities to read and write the Topic.

7.4.4.2 BuiltinParticipantVolatileMessageSecureWriter and BuiltinParticipantVolatileMessageSecureReader

The DDS Security specification defines two new builtin Endpoints: The ***BuiltinParticipantVolatileMessageSecureWriter*** and the ***BuiltinParticipantVolatileMessageSecureReader***. These two endpoints shall be present in compliant implementations of this specification. These endpoints are used to write and read the builtin ***ParticipantVolatileMessageSecure*** `Topic` and shall have the `TopicSecurityAtributes` and `EndpointSecurityAttributes` set as specified in the tables below.

Table 16 – ParticipantVolatileMessageSecure Topic Security Attributes

Attribute	Value
is_read_protected	false
is_write_protected	false
is_discovery_protected	false
is_liveliness_protected	false

Table 17 – ParticipantVolatileMessageSecure Endpoint Security Attributes (Reader and Writer)

Attribute	Value
is_read_protected	false
is_write_protected	false
is_discovery_protected	false
is_liveliness_protected	false
is_submessage_protected	true
is_payload_protected	false
is_key_protected	false

The ***BuiltinParticipantVolatileMessageSecureWriter*** is an RTPS Reliable StatefulWriter (see sub clause 8.4.9.2 of the DDS Interoperability Wire Protocol [2]). The DDS `DataWriter` Qos associated with the `DataWriter` shall be as defined in the table below. Any policies that are not shown in the table shall be set corresponding to the DDS defaults.

Table 18 – Non-default Qos policies for BuiltinParticipantVolatileMessageSecureWriter

DataWriter Qos policy	Policy Value
RELIABILITY	kind= RELIABLE
HISTORY	kind= KEEP_ALL
DURABILITY	kind= VOLATILE

The ***BuiltinParticipantVolatileMessageSecureReader*** is an RTPS Reliable StatefulReader (see sub clause 8.4.11.2 of the DDS Interoperability Wire Protocol [2]). The DDS `DataReader` Qos associated with the `DataReader` shall be as defined in the table below. Any policies that are not shown in the table shall be set corresponding to the DDS defaults.

Table 19 – Non-default Qos policies for BuiltinParticipantVolatileMessageSecureReader

DataReader Qos policy	Policy Value
RELIABILITY	kind= RELIABLE
HISTORY	kind= KEEP_ALL
DURABILITY	kind= VOLATILE

The data type associated with these endpoints is `ParticipantVolatileMessageSecure` defined as:

```
typedef ParticipantVolatileMessageSecure ParticipantGenericMessage;
```

The RTPS `EntityId_t` associated with the ***BuiltinParticipantVolatileMessageSecureWriter*** and ***BuiltinParticipantVolatileMessageSecureReader*** shall be as specified in 7.4.5.

7.4.4.3 Contents of the ParticipantVolatileMessageSecure

The `ParticipantVolatileMessageSecure` is intended as a holder of secure information that is sent point-to-point from a `DomainParticipant` to another.

The ***destination_participant_guid*** shall contain either the value ***GUID_UNKNOWN*** (see sub clause 9.3.1.5 of the DDS Interoperability Wire Protocol [2] or else the `GUID_t` of the destination `DomainParticipant`.

The ***message_identity*** uniquely identifies each individual `ParticipantVolatileMessageSecure`:

- The ***source_guid*** field within the ***message_identity*** shall be set to match the `GUID_t` of the ***BuiltinParticipantVolatileMessageSecureWriter*** that writes the message.
- The ***sequence_number*** field within the ***message_identity*** shall start with the value set to one and be incremented for each different message sent by the ***BuiltinParticipantVolatileMessageSecureWriter***.

The ***related_message_identity*** uniquely identifies another `ParticipantVolatileMessageSecure` that is related to the message being processed. It shall

be set to either the tuple {***GUID_UNKNOWN, 0***} if the message is not related to any other message, or else set to match the ***message_identity*** of the related `ParticipantVolatileMessageSecure.`

The contents ***message_data*** depend on the value of the ***message_class_id*** and are defined in this specification in the sub clause that introduces each one of the defined values of the `GenericMessageClassId`, see 7.4.4.5.

7.4.4.4 Destination of the ParticipantVolatileMessageSecure

If the ***destination_participant_guid*** member is not set to ***GUID_UNKNOWN***, the message written is intended only for the ***BuiltinParticipantVolatileMessageSecureReader*** belonging to the `DomainParticipant` with a matching Participant Key.

This is equivalent to saying that the ***BuiltinParticipantVolatileMessageSecureReader*** has an implied content filter with the logical expression:

"destination_participant_guid == GUID_UNKNOWN

|| destination_participant_guid==BuiltinParticipantVolatileMessageSecureReader.participant.guid"

Implementations of the specification can use this content filter or some other mechanism as long as the resulting behavior is equivalent to having this filter.

If the ***destination_endpoint_guid*** member is not set to ***GUID_UNKNOWN*** the message written targets a specific endpoint within the destination DomainParticipant. The targeted endpoint is the one whose Endpoint Key (DataWriter or DataReader GUID_t) matches the ***destination_endpoint_guid***. This attribute provides a mechanism to specify finer granularity on the intended recipient of a message beyond the granularity provided by the ***destination_participant_guid***.

7.4.4.5 Reserved values of ParticipantVolatileMessageSecure GenericMessageClassId

This specification, including future versions of this specification reserves ***GenericMessageClassId*** values that start with the prefix "`dds.sec.`" (without the quotes) .

The specification defines and uses the following specific values for the ***GenericMessageClassId***:

```
#define GMCLASSID_SECURITY_PARTICIPANT_CRYPTO_TOKENS \
            "dds.sec.participant_crypto_tokens"
#define GMCLASSID_SECURITY_DATAWRITER_CRYPTO_TOKENS  \
            "dds.sec.datawriter_crypto_tokens"
#define GMCLASSID_SECURITY_DATAREADER_CRYPTO_TOKENS  \
            "dds.sec.datareader_crypto_tokens"
```

Additional values of the ***GenericMessageClassId*** may be defined with each plugin implementation.

7.4.4.6 Format of data within ParticipantVolatileMessageSecure

Each value for the `GenericMessageClassId` uses different schema to store data within the generic attributes in the ***message_data***.

7.4.4.6.1 Data for message class GMCLASS_SECURITY_PARTICIPANT_CRYPTO_TOKENS

If `GenericMessageClassId` is `GMCLASSID_SECURITY_PARTICIPANT_CRYPTO_TOKENS`, the ***message_data*** attribute shall contain the `ParticipantCryptoTokenSeq`.

This message is intended to send cryptographic material from one `DomainParticipant` to another when the cryptographic material applies to the whole `DomainParticipant` and not a specific `DataReader` or `DataWriter` within.

The concrete contents of the `ParticipantCryptoTokenSeq` shall be defined by the Cryptographic Plugin (CryptoKeyFactory).

The *destination_participant_guid* shall be set to the `GUID_t` of the destination `DomainParticipant`.

The *destination_endpoint_guid* shall be set to *GUID_UNKNOWN*. This indicates that there is no specific endpoint targeted by this message: It is intended for the whole `DomainParticipant`.

The *source_endpoint_guid* shall be set to *GUID_UNKNOWN*.

7.4.4.6.2　Data for message class GMCLASSID_SECURITY_DATAWRITER_CRYPTO_TOKENS

If `GenericMessageClassId` is `GMCLASSID_SECURITY_DATAWRITER_CRYPTO_TOKENS`, the *message_data* shall contain the `DatawriterCryptoTokenSeq`.

This message is intended to send cryptographic material from one `DataWriter` to a `DataReader` whom it wishes to send information to. The cryptographic material applies to a specific 'sending' `DataWriter` and it is constructed for a specific 'receiving' `DataReader`. This may be used to send the crypto keys used by a `DataWriter` to encrypt data and sign the data it sends to a `DataReader`.

The concrete contents of the `DatawriterCryptoTokenSeq` shall be defined by the Cryptographic Plugin (CryptoKeyFactory).

The *destination_endpoint_guid* shall be set to the `GUID_t` of the `DataReader` that should receive the `CryptoToken` values in the message.

The *source_endpoint_guid* shall be set to the `GUID_t` of the `DataWriter` that will be using the `CryptoToken` values to encode the data it sends to the `DataReader`.

7.4.4.6.3　Data for message class GMCLASSID_SECURITY_DATAREADER_CRYPTO_TOKENS

If `GenericMessageClassId` is `GMCLASSID_SECURITY_DATAWRITER_CRYPTO_TOKENS`, the *message_data* attribute shall contain the `DatareaderCryptoTokenSeq`.

This message is intended to send cryptographic material from one `DataReader` to a `DataWriter` whom it wishes to send information to. The cryptographic material applies to a specific 'sending' `DataReader` and it is constructed for a specific 'receiving' `DataWriter`. This may be used to send the crypto keys used by a `DataReader` to encrypt data and sign the ACKNACK messages it sends to a `DataWriter`.

The concrete contents of the `DatareaderCryptoTokenSeq` shall be defined by the Cryptographic Plugin (CryptoKeyFactory).

The *destination_endpoint_guid* shall be set to the `GUID_t` of the `DataWriter` that should receive the `CryptoToken` values in the message.

The *source_endpoint_guid* shall be set to the `GUID_t` of the `DataReader` that will be using the `CryptoToken` values to encode the data it sends to the `DataWriter`.

7.4.5   Definition of the "Builtin Secure Endpoints"

The complete list of builtin Endpoints that are protected by the security mechanism introduced in the DDS Security specification is: ***SPDPbuiltinParticipantsSecureWriter,*** ***SPDPbuiltinParticipantsSecureReader, SEDPbuiltinPublicationsSecureWriter,*** ***SEDPbuiltinPublicationsSecureReader, SEDPbuiltinSubscriptionsSecureWriter,*** ***SEDPbuiltinSubscriptionsSecureReader, BuiltinParticipantMessageSecureWriter,*** ***BuiltinParticipantMessageSecureReader, BuiltinParticipantVolatileMessageSecureWriter,*** and ***BuiltinParticipantVolatileMessageSecureReader.***

This list shall be referred to as the **builtin secure endpoints**.

7.4.6   Definition of the "Builtin Secure Discovery Endpoints"

The "builtin secure discovery endpoints" is the subset the builtin secure endpoints that are used for discovery. They are: ***SPDPbuiltinParticipantsSecureWriter, SPDPbuiltinParticipantsSecureReader,*** ***SEDPbuiltinPublicationsSecureWriter, SEDPbuiltinPublicationsSecureReader,*** ***SEDPbuiltinSubscriptionsSecureWriter,*** and ***SEDPbuiltinSubscriptionsSecureReader,***

This list shall be referred to as the **builtin secure discovery endpoints**.

7.4.7   Definition of the "Builtin Secure Liveliness Endpoints"

The "builtin secure liveliness endpoints" is the subset the builtin secure endpoints that are used for managing automatic liveliness. They are: ***BuiltinParticipantMessageSecureWriter*** and ***BuiltinParticipantMessageSecureReader***.

This list shall be referred to as the **builtin secure liveliness endpoints**.

7.4.8   Securing the "Builtin Secure Endpoints"

As with application defined Topics, the middleware shall call the operations `get_datawriter_sec_attributes` and `get_datareader_sec_attributes` on the `AccessControl` interface to obtain the `EndpointSecurityAttributes` associated with `DataReader` and `DataWriter` entities on all the "Builtin Secure Endpoints". The specific values of the `EndpointSecurityAttributes` shall be as shown in the Table below:

Table 20 – EndpointSecurityAttributes for all "Builtin Security Endpoints"

Attribute	*DCPSParticipantSecure, DCPSPublicationsSecure, DCPSSubscriptionsSecure*	*DCPSParticipantMessageSecure*	*DCPSParticipantStatelessMessage*	*DCPSParticipantVolatileMessageSecure*
is_read_protected	false	false	false	false
is_write_protected	false	false	false	false
is_discovery_protected	N/A	N/A	N/A	N/A
is_liveliness_protected	N/A	N/A	N/A	N/A
is_submessage_protected	Set to match `ParticipantSecurityAttributes` *is_discovery_protected*	Set to match `ParticipantSecurityAttributes` *is_liveliness_protected*	false	true
is_payload_protected	false	false	false	false
is_key_protected	false	false	false	false

The **false** settings for the *is_read_protected* and *is_write_protected* indicate that these secure builtin endpoints are not protected by the same AccessControl mechanisms as the regular endpoints (i.e. the `AccessControl` plugin is not called). However they are still protected by the access control mechanism imposed by the `DomainParticipant`. That is, if `ParticipantSecurityAttributes` member *is_access_protected* is **true**, then access to the secure builtin topics is protected.

For a description of the `ParticipantSecurityAttributes`, see clause 8.4.2.4.

8 Plugin Architecture

8.1 Introduction

8.1.1 Service Plugin Interface Overview

There are five plugin SPIs: Authentication, Access-Control, Cryptographic, Logging, and Data Tagging.
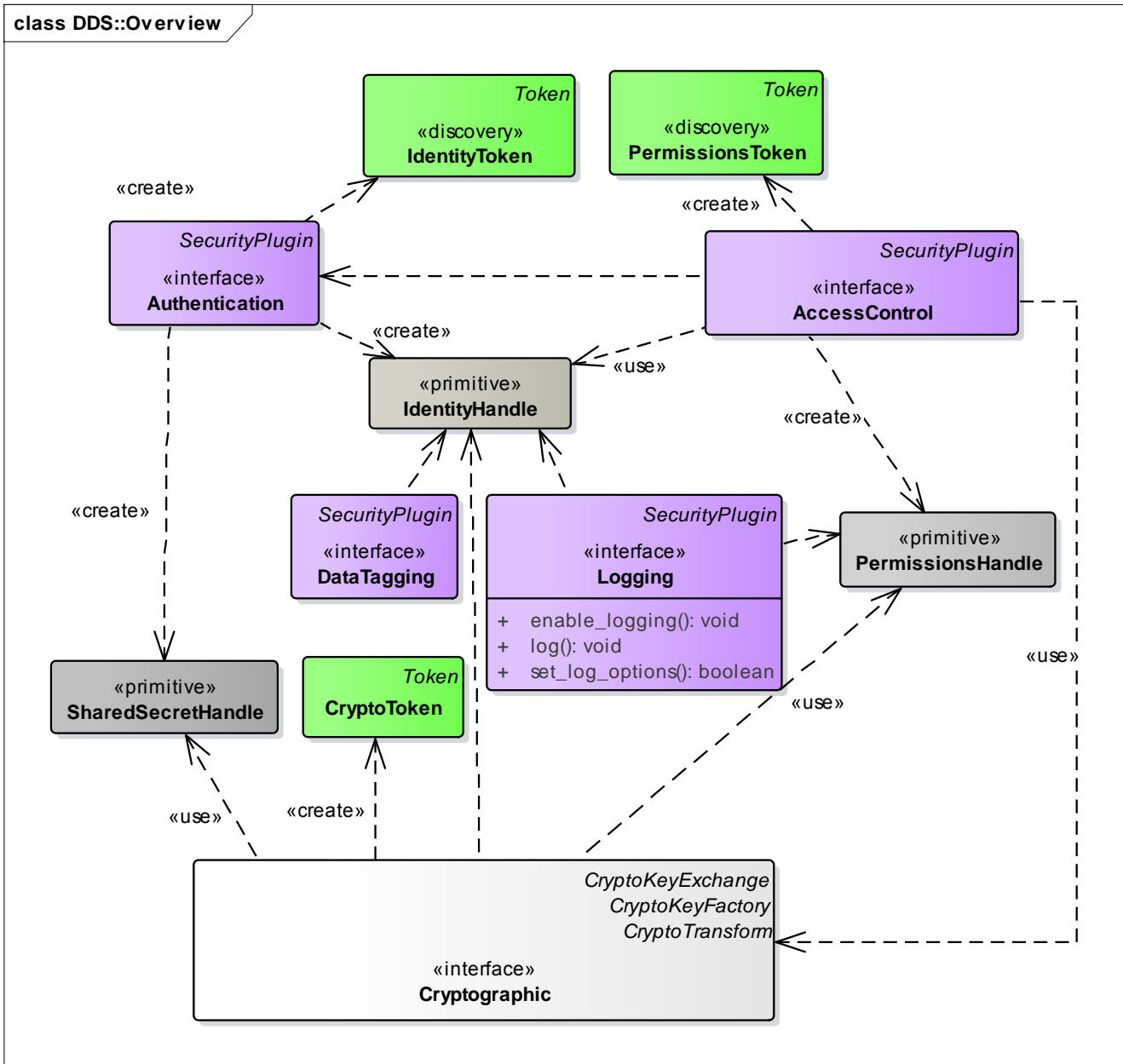
Figure 7 – Plugin Architecture Model

The responsibilities and interactions between these Service Plugins are summarized in the table below and detailed in the sections that follow.

Table 21 – Purpose of each Security Plugin

Service Plugin	*Purpose*	*Interactions*
Authentication	Authenticate the principal that is joining a DDS Domain. Support mutual authentication between participants and establish a shared secret.	The principal may be an application/process or the user associated with that application or process.
AccessControl	Decide whether a principal is allowed to perform a protected operation.	Protected operations include joining a specific DDS domain, creating a Topic, reading a Topic, writing a Topic, etc.
Cryptography	Generate keys. Perform Key Exchange. Perform the encryption and decryption operations. Compute digests, compute and verify Message Authentication Codes. Sign and verify signatures of messages.	This plugin implements 3 complementary interfaces: CryptoKeyFactory, CryptoKeyExchange, and CryptoTransform.
Logging	Log all security relevant events.	This plugin is accessible to all other plugins such that they can log the relevant events.
DataTagging	Add a data tag for each data sample.	

8.1.2 Plugin Instantiation

The Security Plugins shall be configurable separately for each `DomainParticipant` even when multiple DomainParticipants are constructed within the same Operating System Process and share the same Address Space.

A collection of the 5 SPIs intended to be used with the same `DomainParticipant` is referred to as a DDS-Security Plugin Suite.

The mechanism used to instantiate the security Service Plugins and associate them with each `DomainParticipant` is not defined by the DDS-Security specification.

Implementations of this specification may use vendor-specific configurations to facilitate linking the Plugin Suite, including providing dynamic loading and linking facilities as well as initializing the Plugin Suite.

Likewise implementations of this specification may use vendor-specific configurations to bind a Plugin Suite to the `DomainParticipant`. However it is required for the Plugin Suite to be initialized and bound by the time the `DomainParticipant` is enabled. Therefore this process shall complete either during the `DomainParticipantFactory` `create_domain_participant` or else during the `DomainParticipant` `enable` operations defined in [1]. Note that some of the Plugin Suite Authentication and AccessControl operations shall also be called during `create_domain_participant` or during `enable`.

8.2 Common Types

8.2.1 Security Exception

`SecurityException` is a data type used to hold error information. `SecurityException` objects are potentially returned from many of the calls in the Security plugins. They are used to return an error code and message.

Table 22 – SecurityException class

SecurityException	
Attributes	
message	String
code	long
minor_code	long

8.3 Authentication Plugin

The Authentication Plugin SPI defines the types and operations necessary to support the authentication of DDS `DomainParticipants`.

8.3.1 Background (Non-Normative)

Without the security enhancements, any DDS `DomainParticipant` is allowed to join a DDS `Domain` without authenticating. However, in the case of a secure DDS system, every DDS participant will be required to authenticate to avoid data contamination from unauthenticated participants.

The DDS protocol uses its native discovery mechanism to detect when participants enter the DDS `Domain`.

The discovery mechanism that registers participants with the DDS middleware is enhanced with an authentication protocol. For protected DDS Domains a `DomainParticipant` that enables the authentication plugin will only communicate with another `DomainParticipant` that has the authentication plugin enabled.

The plugin SPI is designed to support multiple implementations with varying numbers of message exchanges. The message exchanges may be used by two DomainParticipant entities to challenge each other so that their identity can be authenticated. Often a shared secret is also derived from a successful authentication message exchange. The shared secret can be used to exchange cryptographic materal in support of encryption and message authentication.

8.3.2 Authentication Plugin Model

The Authentication Plugin model is presented in the figure below.
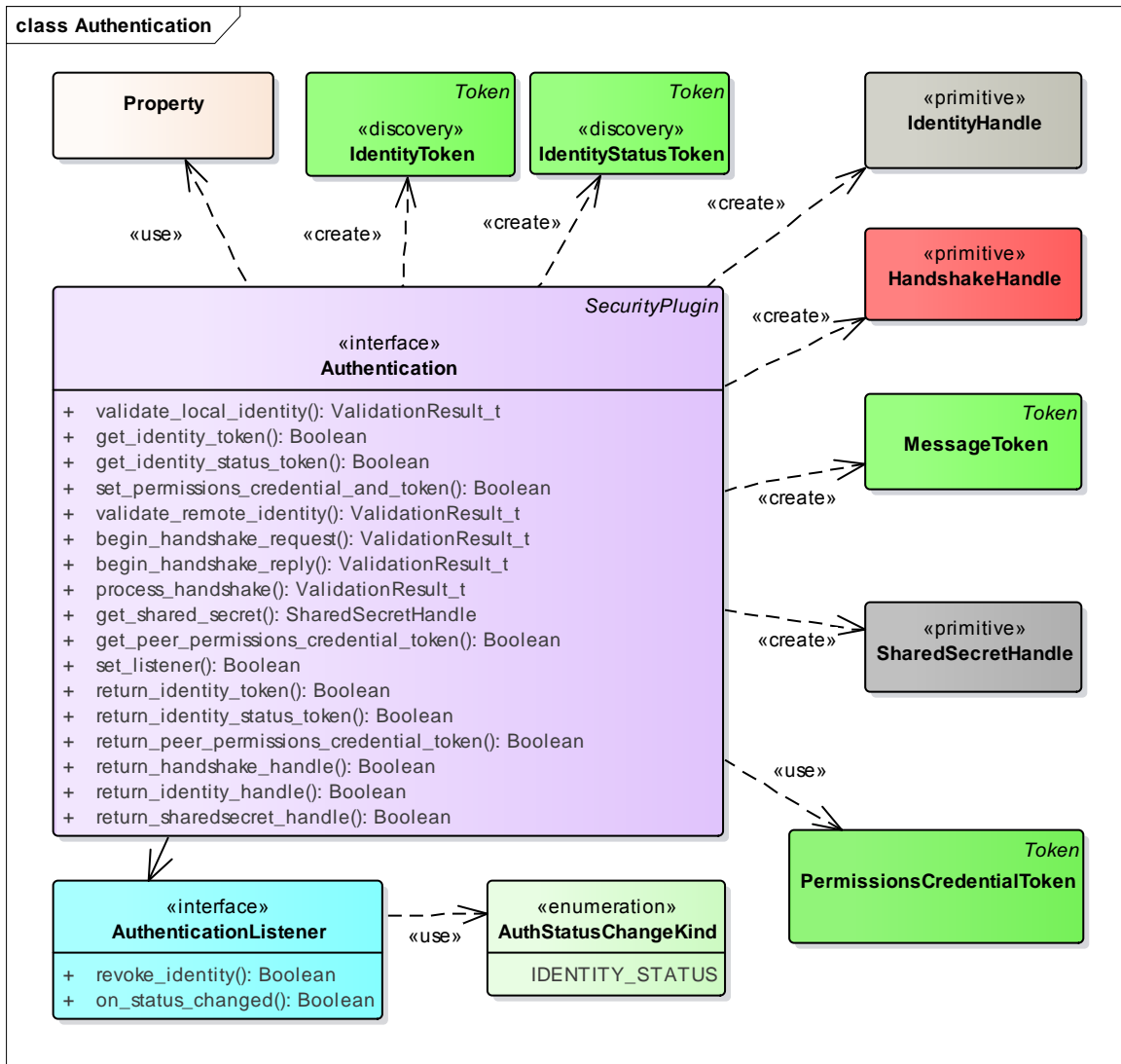
Figure 8 – Authentication plugin model

8.3.2.1 IdentityToken

An `IdentityToken` contains summary information on the identity of a `DomainParticipant` in a manner that can be externalized and propagated via DDS discovery. The specific content of the `IdentityToken` shall be defined by each `Authentication` plugin specialization. The intent is to provide only summary information on the permissions or derived information such as a hash.

8.3.2.2 IdentityStatusToken

An `IdentityStatusToken` contains authentication information of a `DomainParticipant` in a manner that can be externalized and propagated via the ***DCPSParticipantSecure*** builtin `Topic`. The specific content of the `IdentityStatusToken` shall be defined by each `Authentication` plugin. The intent is to provide a mechanism that can be used to securely send information to other participants that are already mutually authenticated. It could be used, for example, to provide an updated certificate in case the current one has expired.

The information shall be retrieved from the `Authentication` plugin by calling the operation `get_identity_status_token`. And included in the **DCPSParticipantSecure** builtin `Topic`. The `Authentication` plugin can use the operation `on_status_changed` on the `AuthenticationListener` to notify that there is an updated `IdentityStatusToken`.

8.3.2.3   IdentityHandle

An `IdentityHandle` is an opaque local reference to internal state within the `Authentication` plugin, which uniquely identifies a `DomainParticipant`. It is understood only by the `Authentication` plugin and references the authentication state of the `DomainParticipant`. This object is returned by the `Authentication` plugin as part of the validation of the identity of a `DomainParticipant` and is used whenever a client of the `Authentication` plugin needs to refer to the identity of a previously identified `DomainParticipant`.

8.3.2.4   HandshakeHandle

A `HandshakeHandle` is an opaque local reference used to refer to the internal state of a possible mutual authentication or handshake protocol.

8.3.2.5   AuthRequestMessageToken

The `AuthRequestMessageToken` encodes plugin-specific information that the `Authentication` plugins associated with two `DomainParticipant` entities exchange to bootstrap the mutual authentication handshake. The `AuthRequestMessageToken` is understood only by the `AuthenticationPlugin` implementations on either side of the handshake. The `AuthRequestMessageToken` is sent and received by the DDS implementation under the direction of the `AuthenticationPlugins`.

The `AuthRequestMessageToken` has *class_id* set to GMCLASSID_SECURITY_AUTH_REQUEST (see 7.4.3.5).

8.3.2.6   HandshakeMessageToken

A `HandshakeMessageToken` encodes plugin-specific information that the Authentication plugins associated with two DomainParticipant entities exchange as part of the mutual authentication handshake. The `HandshakeMessageToken` is understood only by the `AuthenticationPlugin` implementations on either side of the handshake. The `HandshakeMessageToken` is sent and received by the DDS implementation under the direction of the `AuthenticationPlugins`.

The `HandshakeMessageToken` has *class_id* set to GMCLASSID_SECURITY_AUTH_HANDSHAKE (see 7.4.3.5).

8.3.2.7   AuthenticatedPeerCredentialToken

An `AuthenticatedPeerCredentialToken` encodes plugin-specific information that the Authentication plugin obtains from a remote DomainParticipant during the authentication process that is of interest to the `AccessControlPlugin`. This information is accessible via the operation `get_authenticated_peer_credential_token`.

8.3.2.8 SharedSecretHandle

A `SharedSecretHandle` is an opaque local reference to internal state within the `AuthenticationPlugin` containing a secret that is shared between the `AuthenticationPlugin` implementation and the peer `AuthenticationPlugin` implementation associated with a remote `DomainParticipant`. It is understood only by the two `AuthenticationPlugin` implementations that share the secret. The shared secret is used to encode Tokens, such as the `CryptoToken`, such that they can be exchanged between the two `DomainParticipants` in a secure manner.

8.3.2.9 Authentication

This interface is the starting point for all the security mechanisms. When a `DomainParticipant` is either locally created or discovered, it needs to be authenticated in order to be able to communicate in a DDS `Domain`.

The interaction between the DDS implementation and the Authentication plugin has been designed in a flexible manner so it is possible to support various authentication mechanisms, including those that require a handshake and/or perform mutual authentication between participants. It also supports establishing a shared secret. This interaction is described in the state machine illustrated in the figure below.
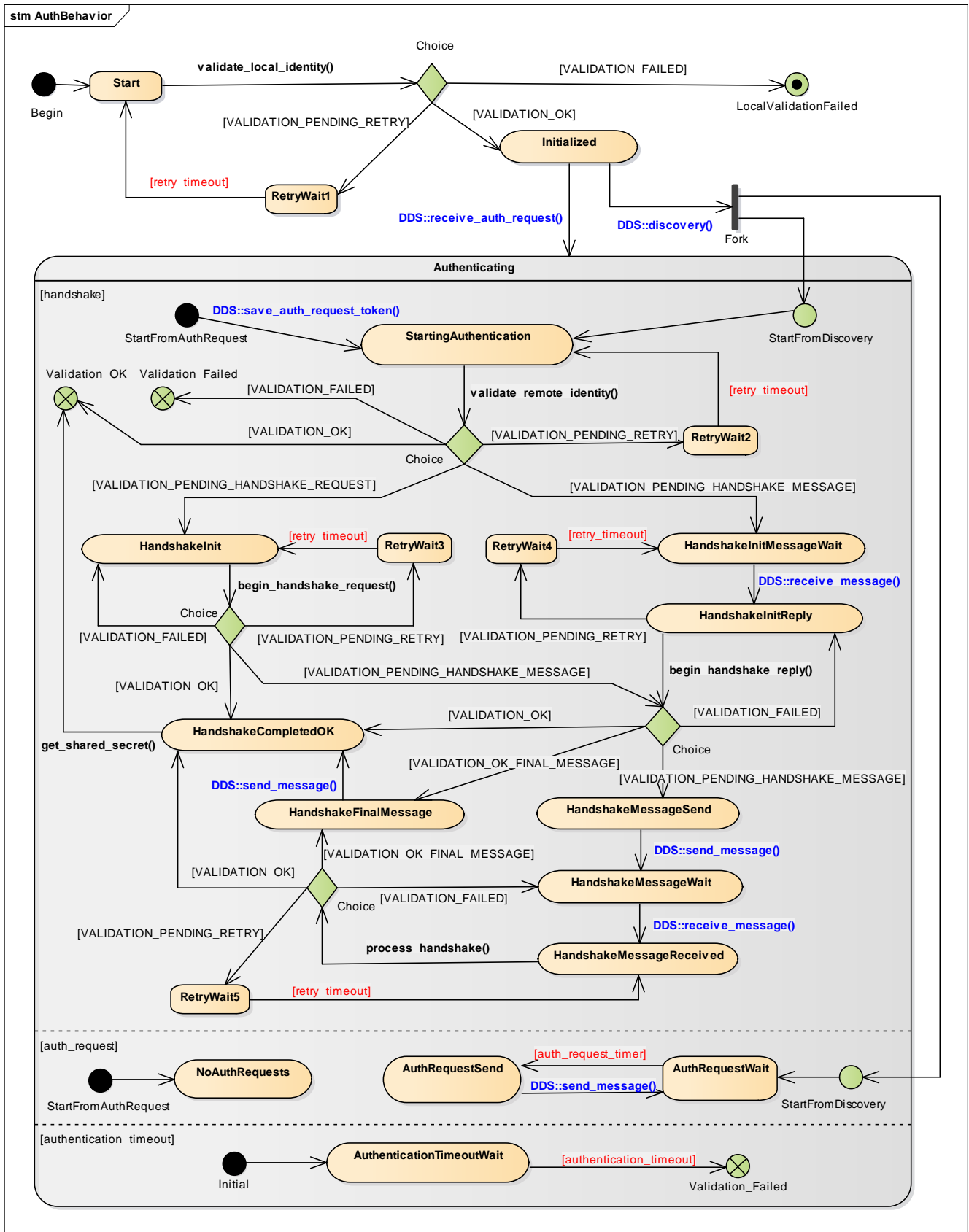
Figure 9 – Authentication plugin interaction state machine

8.3.2.9.1   Reliability of the Authentication Handshake

In order to be sufficiently robust to avert sequence number attacks (7.4.3.1), the Authentication Handshake uses the ***BuiltinParticipantStatelessMessageWriter*** and ***BuiltinParticipantStatelessMessageReader*** endpoints (7.4.3) with `GenericMessageClassId` set to GMCLASSID_SECURITY_AUTH_REQUEST or GMCLASSID_SECURITY_AUTH_HANDSHAKE (7.4.3.5). These stateless endpoints send messages best-effort without paying attention to any sequence number information to remove duplicates or attempt ordered delivery. Despite this, the Authentication Handshake needs to be able to withstand the message loss that may occur on the network.

In order to operate robustly in the presence of message loss and sequence number attacks DDS Security implementations shall follow the rules below:

1. The DDS security implementation shall pass to the AuthenticationPlugin any message received by the ***BuiltinParticipantStatelessMessageReader*** that has a `GenericMessageClassId` set to GMCLASSID_SECURITY_AUTH_REQUEST or GMCLASSID_SECURITY_AUTH_HANDSHAKE.
2. Any time the state-machine indicates that a message shall be sent using the ***BuiltinParticipantStatelessMessageWriter*** and a reply message needs to be received by the ***BuiltinParticipantStatelessMessageReader***, the DDS implementation shall cache the message that was sent and set a timer. If a correct reply message is not received when the timer expires, the state-machine shall send the same message again. This process shall be repeated multiple times until a correct message is received.
3. Whenever a message is sent using the ***BuiltinParticipantStatelessMessageWriter,*** a reply message is received by the ***BuiltinParticipantStatelessMessageReader***. The reply is then passed to the AuthenticationPlugin. If the plugin operation returns VALIDATION_NOT_OK, the implementation transitions back to the previous state that caused the message to be sent and resends the same message.

Rule #2 makes authentication robust to message loss.

Rule #3 makes authentication robust to an attacker trying to disrupt an authentication exchange by sending bad replies.

Example application of rule #2: Assume the DDS implementation transitioned to the *HandshakeMessageSend* state, sent the message M1 and is now in the *HandshakeMessageWait* state waiting for the reply. If no reply is received within an implementation-specific retry-time, the same message M1 shall be sent again and the process repeated until either a reply is received or an implementation-specific timeout elapses (or a maximum number of retries is reached).

Example application of rule #3: Assume the DDS implementation transitioned to the *HandshakeMessageSend* state, sent the message M2, transitions to *HandshakeMessageWait*, receives the reply, transitions to *HandshakeMessageReceived*, calls process_handshake() and the operation returns VALIDATION_NOT_OK. In this situation the DDS implementation shall transition back to *HandshakeMessageSend* and resent M2 again.

8.3.2.10   Unauthenticated DomainParticipant entities

The term "Unauthenticated" `DomainParticipant` entity refers to a discovered `DomainParticipant` that cannot be authenticated by the Authentication plugin. This can be either

because they lack support for the `Authentication` plugin being used, have incompatible plugins, or simply fail the authentication protocol.

8.3.2.11 Authentication plugin interface

The `Authentication` plugin shall have the operations shown in the table below.

Table 23 – Authentication plugin interface

Authentication		
No Attributes		
Operations		
`validate_local_identity`		`ValidationResult_t`
	`out:` `local_identity_handle`	`IdentityHandle`
	`out:` `adjusted_participant_guid`	`GUID_t`
	`domain_id`	`DomainId_t`
	`participant_qos`	`DomainParticipantQos`
	`candidate_participant_guid`	`GUID_t`
	`out: exception`	`SecurityException`
`get_identity_token`		`Boolean`
	`out: identity_token`	`IdentityToken`
	`handle`	`IdentityHandle`
	`out: exception`	`SecurityException`
`get_identity_status_token`		`Boolean`
	`out:` `identity_status_token`	`IdentityStatusToken`
	`handle`	`IdentityHandle`
	`out: exception`	`SecurityException`

set_permissions_credential_and_token		Boolean
	handle	IdentityHandle
	permissions_credential_token	PermissionsCredentialToken
	permissions_token	PermissionsToken
	out: exception	SecurityException
validate_remote_identity		ValidationResult_t
	out: remote_identity_handle	IdentityHandle
	out: local_auth_request_token	AuthRequestMessageToken
	remote_auth_request_token	AuthRequestMessageToken
	local_identity_handle	IdentityHandle
	remote_identity_token	IdentityToken
	remote_participant_guid	GUID_t
	out: exception	SecurityException
begin_handshake_request		ValidationResult_t
	out: handshake_handle	HandshakeHandle
	out: handshake_message	HandshakeMessageToken
	initiator_identity_handle	IdentityHandle
	replier_identity_handle	IdentityHandle
	serialized_local_participant_data	octet[]
	out: exception	SecurityException

begin_handshake_rep ly		ValidationResult_t
	out: handshake_handle	HandshakeHandle
	out: handshake_message_out	HandshakeMessageToken
	handshake_message_in	HandshakeMessageToken
	initiator_identity_handle	IdentityHandle
	replier_identity_handle	IdentityHandle
	serialized_local_particip ant_data	octet[]
	out: exception	SecurityException
process_handshake		ValidationResult_t
	out: handshake_message_out	HandshakeMessageToken
	handshake_message_in	HandshakeMessageToken
	handshake_handle	HandshakeHandle
	out: exception	SecurityException
get_shared_secret		SharedSecretHandle
	handshake_handle	HandshakeHandle
	out: exception	SecurityException
get_authenticated_p eer_credential_toke n		Boolean
	out: peer_credential_token	AuthenticatedPeerCre dentialToken
	handshake_handle	HandshakeHandle
	out: exception	SecurityException
set_listener		Boolean
	listener	AuthenticationListen er
	out: exception	SecurityException

`return_identity_tok` `en`		`Boolean`
	`token`	`IdentityToken`
	`out: exception`	`SecurityException`
`return_identity_sta` `tus_token`		`Boolean`
	`token`	`IdentityStatusToken`
	`out: exception`	`SecurityException`

`return_authenticate`
`d_peer_credential_t`
`oken`
		`Boolean`

	`peer_credential_token`	`AuthenticatedPeerCre` `dentialToken`
	`out: exception`	`SecurityException`
`return_handshake_ha` `ndle`		`Boolean`
	`handshake_handle`	`HandshakeHandle`
	`out: exception`	`SecurityException`
`return_identity_han` `dle`		`Boolean`
	`identity_handle`	`IdentityHandle`
	`out: exception`	`SecurityException`
`return_sharedsecret` `_handle`		`Boolean`
	`sharedsecret_handle`	`SharedSecretHandle`
	`out: exception`	`SecurityException`

8.3.2.11.1 Type: ValidationResult_t

Enumerates the possible return values of the `validate_local_identity` and `validate_remote_identity` operations.

Table 24 – Values for ValidationResult_t

ValidationResult_t	
`VALIDATION_OK`	Indicates the validation has succeeded
`VALIDATION_FAILED`	Indicates the validation has failed
`VALIDATION_PENDING_` `RETRY`	Indicates that validation is still proceeding. The operation shall be retried at a later point in time.

VALIDATION_PENDING_ HANDSHAKE_REQUEST	Indicates that validation of the submitted `IdentityToken` requires sending a handshake message. The DDS Implementation shall call the operation `begin_handshake_request` and send the `HandshakeMessageToken` obtained from this call using the ***BuiltinParticipantMessageWriter*** with `GenericMessageClassId` set to GMCLASSID_SECURITY_AUTH_HANDSHAKE.
VALIDATION_PENDING_ HANDSHAKE_MESSAGE	Indicates that validation is still pending. The DDS Implementation shall wait for a message on the ***BuiltinParticipantMessageReader*** and, once this is received, call `process_handshake` to pass the information received in that message.
VALIDATION_OK_FINAL _MESSAGE	Indicates that validation has succeeded but the DDS Implementation shall send a final message using the ***BuiltinParticipantMessageWriter*** with `GenericMessageClassId` set to GMCLASSID_SECURITY_AUTH_HANDSHAKE.

8.3.2.11.2 Operation: `validate_local_identity`

Validates the identity of the local `DomainParticipant`. The operation returns as an output parameter the `IdentityHandle`, which can be used to locally identify the local Participant to the `Authentication Plugin`.

In addition to validating the identity, this operation also returns the `DomainParticipant GUID_t` that shall be used by the DDS implementation to uniquely identify the `DomainParticipant` on the network.

This operation shall be called before the `DomainParticipant` is enabled. It shall be called either by the implementation of `DomainParticipantFactory create_domain_participant` or `DomainParticipant enable` [1].

If an error occurs, this method shall return VALIDATION_FAILED and fill the `SecurityException`.

The method shall return either VALIDATION_OK if the validation succeeds, or VALIDATION_FAILED if it fails, or VALIDATION_PENDING_RETRY if the verification has not finished.

If VALIDATION_PENDING_RETRY has been returned, the operation shall be called again after a configurable delay to check the status of verification. This shall continue until the operation returns either VALIDATION_OK (if the validation succeeds), or VALIDATION_FAILED. This approach allows non-blocking interactions with services whose verification may require invoking remote services.

Parameter (out) local_identity_handle: A handle that can be used to locally refer to the Authenticated Participant in subsequent interactions with the `Authentication` plugin. The nature of the handle is specific to each `Authentication` plugin implementation. The handle will only be meaningful if the operation returns VALIDATION_OK.

Parameter (out) adjusted_participant_guid: The `GUID_t` that the DDS implementation shall use to uniquely identify the `DomainParticipant` on the network. The returned *adjusted_participant_guid* shall be the one that eventually appears in the *participant_guid* attribute of the `ParticipantBuiltinTopicData` sent via discovery.

Parameter domain_id: The DDS Domain Id of the `DomainParticipant`.

Parameter participant_qos: The `DomainParticipantQos` of the `DomainParticipant`.

Parameter candidate_participant_guid: The `GUID_t` that the DDS implementation would have used to uniquely identify the `DomainParticipant` if the Security plugins were not enabled.

Parameter exception: A `SecurityException` object.

Return: The operation shall return

- VALIDATION_OK if the validation was successful.
- VALIDATION_FAILED if it failed.
- VALIDATION_PENDING_RETRY if verification has not completed and the operation should be retried later.

8.3.2.11.3 Operation: `validate_remote_identity`

Initiates the process of validating the identity of the discovered remote `DomainParticipant`, represented as an `IdentityToken` object. The operation returns the `ValidationResult_t` indicating whether the validation succeeded, failed, or is pending a handshake. If the validation succeeds, an `IdentityHandle` object is returned, which can be used to locally identify the remote `DomainParticipant` to the `Authentication` plugin.

If the validation can be performed with the information passed and succeeds, the operation shall return VALIDATION_OK. If it can be performed with the information passed and it fails, it shall return VALIDATION_FAILED.

The validation of a remote participant might require the remote participant to perform a handshake. In this situation, the `validate_remote_identity` operation shall return VALIDATION_PENDING_HANDSHAKE_REQUEST or VALIDATION_PENDING_HANDSHAKE_MESSAGE.

If the operation returns VALIDATION_PENDING_HANDSHAKE_REQUEST, then the DDS implementation shall call the operation `begin_handshake_request` to continue the validation process.

If the operation returns VALIDATION_PENDING_HANDSHAKE_MESSAGE, then the DDS implementation shall wait until it receives a `ParticipantStatelessMessage` from the remote participant identified by the *remote_participant_guid* using the contents described in 8.3.2.11.5 and then call the operation `begin_handshake_reply`.

If an error occurs, this method shall return VALIDATION_FAILED and fill the `SecurityException`.

Parameter (out) remote_identity_handle: A handle that can be used to locally refer to the remote Authenticated Participant in subsequent interactions with the `AuthenticationPlugin`. The nature

of the **remote_identity_handle** is specific to each `AuthenticationPlugin` implementation. The handle will only be provided if the operation returns something other than VALIDATION_FAILED.

Parameter (out) local_auth_request_token: An `AuthRequestMessageToken` to be sent using the ***BuiltinParticipantStatelessMessageWriter***. The contents shall be specified by each plugin implementation. If the returned token is `TokenNIL` (see 7.2.4.3), the `AuthRequestMessageToken` shall not be sent.

Parameter remote_auth_request_token: The `AuthRequestMessageToken` received from the remote `DomainParticipant` that caused the authentication to begin. This token shall be NIL if the authentication was not initiated by the reception of an `AuthRequestMessageToken`.

Parameter remote_identity_token: A token received as part of `ParticipantBuiltinTopicData,` representing the identity of the remote `DomainParticipant`.

Parameter remote_participant_guid: `GUID_t` uniquely identifying the remote participant.

Parameter exception: A SecurityException object.

Return: The operation shall return:

- VALIDATION_OK if the validation was successful.
- VALIDATION_FAILED if it failed.
- VALIDATION_PENDING_HANDSHAKE_REQUEST if validation has not completed. If this is returned, the DDS implementation shall call `begin_handshake_request`, to continue the validation.
- VALIDATION_PENDING_HANDSHAKE_MESSAGE if validation has not completed. If this is returned, the DDS implementation shall wait for a message on the ***BuiltinParticipantMessageReader*** with the ***message_identity*** containing a ***source_guid*** that matches the ***remote_participant_guid*** and a ***message_class_id*** set to GMCLASSID_SECURITY_AUTH_HANDSHAKE.
- VALIDATION_PENDING RETRY if the validation has not completed. If this is returned, the operation should be called again at a later point in time to check the validation status.

8.3.2.11.4 Operation: `begin_handshake_request`

This operation is used to initiate a handshake. It shall be called by the DDS middleware solely as a result of having a previous call to `validate_remote_identity` returning VALIDATION_PENDING_HANDSHAKE_REQUEST.

This operation returns a `HandshakeMessageToken` that shall be used to send a handshake to the remote participant identified by the ***replier_identity_handle***.

The contents of the `HandshakeMessageToken` are specified by the plugin implementation.

If an error occurs, this method shall return VALIDATION_FAILED and fill the `SecurityException`.

Parameter (out) handshake_handle: A handle returned by the `Authentication` plugin used to keep the state of the handshake. It is passed to other operations in the `Authentication` plugin.

Parameter (out) handshake_message_token: A `HandshakeMessageToken` to be sent using the *BuiltinParticipantMessageWriter*. The contents shall be specified by each plugin implementation.

Parameter initiator_identity_handle: Handle to the local participant that originated the handshake.

Parameter replier_identity_handle: Handle to the remote participant whose identity is being validated.

Parameter serialized_local_participant_data: CDR Big Endian Serialization for the `ParticipantBuiltInTopicDataSecure` object associated with the local `DomainParticipant`.

Parameter exception: A `SecurityException` object.

Return: The operation shall return:

- VALIDATION_OK if the validation was successful.
- VALIDATION_FAILED if it failed.
- VALIDATION_PENDING_HANDSHAKE_MESSAGE if validation has not completed. If this is returned, the DDS implementation shall send the *handshake_message_out* using the *BuiltinParticipantMessageWriter* and then wait for the reply message on the *BuiltinParticipantMessageReader*. The DDS implementation shall set the `ParticipantStatelessMessage participantGuidPrefix` message_class_id to GMCLASSID_SECURITY_AUTH_HANDSHAKE and fill the *message_data* with the *handshake_message* `HandshakeMessageToken` and set the *destination_participant_guid* to match the DDS `GUID_t` of the destination `DomainParticipant`. When the reply message is received the DDS implementation shall call the operation `begin_handshake_reply`, to continue the validation.
- VALIDATION_OK_FINAL_MESSAGE if the validation succeeded. If this is returned, the DDS implementation shall send the returned *handshake_message* using the *BuiltinParticipantMessageReader*.
- VALIDATION_PENDING RETRY if the validation has not completed. If this is returned, the DDS implementation shall call the operation again at a later point in time to check the validation status.

In the cases where the return code indicates that a message shall be sent using the *BuiltinParticipantMessageWriter*, the DDS implementation shall set the `ParticipantStatelessMessage` as follows:

- The *message_class_id* shall be set to `GMCLASSID_SECURITY_AUTH_HANDSHAKE`.
- The *destination_participant_guid* shall be set to match the DDS `GUID_t` of the destination `DomainParticipant`.
- The *message_identity* shall be set to have the *source_guid* matching the DDS `GUID_t` of the `DomainParticipant` that is sending the message and the *sequence_number* to the value in the previous message sent by the *BuiltinParticipantMessageWriter*, incremented by one.
- The *related_message_identity* shall be set with *source_guid* as *GUID_UNKNOWN* and *sequence_number* to zero.
- The *message_data* shall be filled with the *handshake_message* `HandshakeMessageToken`.

8.3.2.11.5 Operation: `begin_handshake_reply`

This operation shall be invoked by the DDS implementation in reaction to the reception of the initial handshake message that originated on a `DomainParticipant` that called the `begin_handshake_request` operation. It shall be called by the DDS implementation solely as a result of having a previous call to `validate_remote_identity` returns VALIDATION_PENDING_HANDSHAKE_MESSAGE and having received a message on the ***BuiltinParticipantMessageReader*** with attributes set as follows:

- ***message_class_id*** `GMCLASSID_SECURITY_AUTH_HANDSHAKE`
- ***message_identity source_guid*** matching the `GUID_t` of the `DomainParticipant` associated with the ***initiator_identity_handle***
- ***destination_participant_guid*** matching the `GUID_t` of the receiving `DomainParticipant`

This operation generates a ***handshake_message_out*** in response to a received ***handshake_message_in***. Depending on the return value of the operation, the DDS implementation shall send the ***handshake_message_out*** using the ***BuiltinParticipantMessageWriter*** to the participant identified by the ***initiator_identity_handle***.

The contents of the ***handshake_message_out*** `HandshakeMessageToken` are specified by the plugin implementation.

If an error occurs, this method shall return VALIDATION_FAILED and fill the `SecurityException`.

Parameter (out) handshake_handle: A handle returned by the Authentication Plugin used to keep the state of the handshake. It is passed to other operations in the Plugin.

Parameter (out) handshake_message_out: A `HandshakeMessageToken` containing a message to be sent using the ***BuiltinParticipantMessageWriter***. The contents shall be specified by each plugin implementation.

Parameter handshake_message_in: A `HandshakeMessageToken` containing a message received from the ***BuiltinParticipantMessageReader***. The contents shall be specified by each plugin implementation.

Parameter initiator_identity_handle: Handle to the remote participant that originated the handshake.

Parameter replier_identity_handle: Handle to the local participant that is initiating the handshake response.

Parameter serialized_local_participant_data: CDR Big Endian Serialization for the `ParticipantBuiltInTopicDataSecure` object associated with the local `DomainParticipant`.

Parameter exception: A `SecurityException` object.

Return: The operation shall return:

- VALIDATION_OK if the validation was successful.
- VALIDATION_FAILED if it failed.
- VALIDATION_PENDING_HANDSHAKE_MESSAGE if validation has not completed. If this is returned, the DDS implementation shall send the ***handshake_message_out*** using the

BuiltinParticipantMessageWriter and then wait for a reply message on the ***BuiltinParticipantMessageReader*** from that remote `DomainParticipant`.

- VALIDATION_OK_FINAL_MESSAGE if the validation succeeded. If this is returned, the DDS implementation shall send the returned ***handshake_message_out*** using the ***BuiltinParticipantMessageWriter***.
- VALIDATION_PENDING RETRY if the validation has not completed. If this is returned, the DDS implementation shall call the operation again at a later point in time to check the validation status.

In cases where the return code indicates that a message shall be sent using the ***BuiltinParticipantMessageWriter***, the DDS implementation shall set the `ParticipantStatelessMessage` as follows:

- The ***message_class_id*** shall be set to `GMCLASSID_SECURITY_AUTH_HANDSHAKE`.
- The ***destination_participant_guid*** shall be set to match the DDS `GUID_t` of the destination `DomainParticipant`.
- The ***message_identity*** shall be set to have the ***source_guid*** matching the DDS `GUID_t` of the `DomainParticipant` that is sending the message and the ***sequence_number*** to the value in the previous message sent by the ***BuiltinParticipantMessageWriter***, incremented by one.
- The ***related_message_identity*** shall be set to match the ***message_identity*** of the `ParticipantStatelessMessage` received that triggered the execution of the `begin_handshake_reply` operation.
- The ***message_data*** shall be filled with the ***handshake_message_out*** `HandshakeMessageToken`.

8.3.2.11.6 Operation: `process_handshake`

This operation is used to continue a handshake. It shall be called by the DDS middleware solely as a result of having a previous call to ***begin_handshake_request*** or ***begin_handshake_reply*** that returned VALIDATION_PENDING_HANDSHAKE_MESSAGE and having also received a `ParticipantStatelessMessage` on the ***BuiltinParticipantMessageReader*** with attributes set as follows:

- ***message_class_id*** `GMCLASSID_SECURITY_AUTH_HANDSHAKE`
- ***message_identity source_guid*** matching the `GUID_t` of the peer `DomainParticipant` associated with the ***handshake_handle***
- ***related_message_identity*** matching the ***message_identity*** of the last `ParticipantStatelessMessage` sent to the peer DomainParticipant associated with the ***handshake_handle***.
- ***destination_participant_guid*** matching the `GUID_t` of the receiving `DomainParticipant`.

This operation generates a ***handshake_message_out*** `HandshakeMessageToken` in response to a received ***handshake_message_in*** `HandshakeMessageToken`. Depending on the return value of the function the DDS implementation shall send the ***handshake_message_out*** using the ***BuiltinParticipantMessageWriter*** to the peer participant identified by the ***handshake_handle***.

The contents of the ***handshake_message_out*** `HandshakeMessageToken` are specified by the plugin implementation.

If an error occurs, this method shall return VALIDATION_FAILED and fill the `SecurityException`.

Parameter (out) handshake_message_out: A `HandshakeMessageToken` containing the *message_data* that should be placed in a `ParticipantStatelessMessage` to be sent using the *BuiltinParticipantMessageWriter*. The contents shall be specified by each plugin implementation.

Parameter handshake_message_in: The `HandshakeMessageToken` contained in the *message_data* attribute of the `ParticipantStatelessMessage` received. The interpretation of the contents shall be specified by each plugin implementation.

Parameter handshake_handle: Handle returned by a corresponding previous call to *begin_handshake_request* or *begin_handshake_reply*.

Parameter exception: A `SecurityException` object.

Return: The operation shall return:

- VALIDATION_OK if the validation was successful.
- VALIDATION_FAILED if it failed.
- VALIDATION_PENDING_HANDSHAKE_MESSAGE if validation has not completed. If this is returned, the DDS implementation shall send a `ParticipantStatelessMessage` continuing the returned *handshake_message_out* using the *BuiltinParticipantMessageWriter* and then wait for a reply message on the *BuiltinParticipantMessageReader* from that remote `DomainParticipant`.
- VALIDATION_OK_FINAL_MESSAGE if the validation succeeded. If this is returned, the DDS implementation shall send a `ParticipantStatelessMessage` containing the returned *handshake_message_out* using the *BuiltinParticipantMessageWriter* but not wait for any replies.
- VALIDATION_PENDING RETRY if the validation has not completed. If this is returned, the DDS implementation shall call the operation again at a later point in time to check the validation status.

In the cases where the return code indicates that a `ParticipantStatelessMessage` shall be sent using the *BuiltinParticipantMessageWriter* the DDS implementation shall set the fields of the `ParticipantStatelessMessage` as follows:

- The *message_class_id* shall be set to `GMCLASSID_SECURITY_AUTH_HANDSHAKE`.
- The *destination_participant_guid* shall be set to match the DDS `GUID_t` of the destination `DomainParticipant`.
- The *message_identity* shall be set to have the *source_guid* matching the DDS `GUID_t` of the `DomainParticipant` that is sending the message and the *sequence_number* to the value in the previous message sent by the *BuiltinParticipantMessageWriter*, incremented by one.
- The *related_message_identity* shall be set to match the *message_identity* of the `ParticipantStatelessMessage` received that triggered the execution of the `begin_handshake_reply` operation.
- The *message_data* shall be filled with the *handshake_message_out* `HandshakeMessageToken`.

8.3.2.11.7 Operation: get_shared_secret

Retrieves the `SharedSecretHandle` resulting with a successfully completed handshake.

This operation shall be called by the DDS middleware on each `HandshakeHandle` after the handshake that uses that handle completes successfully, that is after the last 'handshake' operation called on that handle (`begin_handshake_request`, `begin_handshake_reply`, or `process_handshake`) returns VALIDATION_OK or VALIDATION_OK_FINAL_MESSAGE.

The retrieved `SharedSecretHandle` shall be used by the DDS middleware in conjunction with the `CryptoKeyExchange` interface of the `Cryptographic` Plugin to exchange cryptographic key material with other `DomainParticipant` entities.

If an error occurs, this method shall return the `NILHandle` and fill the `SecurityException`.

Parameter handshake_handle: Handle returned by a corresponding previous call to *begin_handshake_request* or *begin_handshake_reply*, which has successfully completed the handshake operations.

Parameter exception: A `SecurityException` object.

8.3.2.11.8 Operation: get_authenticated_peer_ credential_token

Retrieves the `AuthenticatedPeerCredentialToken` resulting with a successfully completed authentication of a discovered `DomainParticipant`.

This operation shall be called by the DDS middleware on each `HandshakeHandle` after the handshake that uses that handle completes successfully, that is after the last 'handshake' operation called on that handle (`begin_handshake_request`, `begin_handshake_reply`, or `process_handshake`) returns VALIDATION_OK or VALIDATION_OK_FINAL_MESSAGE.

If an error occurs, this method shall return false and fill the `SecurityException`.

Parameter peer_credential_token (out): A placeholder for the returned `AuthenticatedPeerCredentialToken`.

Parameter handshake_handle: `HandshakeHandle` returned by a corresponding previous call to `begin_handshake_request` or `begin_handshake_reply`, which has successfully completed the handshake operations.

Parameter exception: A `SecurityException` object.

8.3.2.11.9 Operation: `get_identity_token`

Retrieves an `IdentityToken` used to represent on the network the identity of the `DomainParticipant` identified by the specified `IdentityHandle`.

Parameter identity_token (out): The returned `IdentityToken`.

Parameter handle: The handle used to locally identify the `DomainParticipant` for which an `IdentityToken` is desired. The handle must have been returned by a successful call to `validate_local_identity`, otherwise the operation shall return false and fill the `SecurityException`.

Parameter exception: A `SecurityException` object.

Return: If an error occurs, this method shall return `false` and fill the `SecurityException`. otherwise it shall return the `IdentityToken`.

8.3.2.11.10Operation: `get_identity_status_token`

Retrieves an `AuthenticationToken` used to represent on the network the authentication state of the `DomainParticipant` identified by the specified `IdentityHandle`.

Parameter identity_token (out): The returned `IdentityStatusToken`.

Parameter handle: The handle used to locally identify the `DomainParticipant` for which an `IdentityStatusToken` is desired. The handle must have been returned by a successful call to `validate_local_identity`, otherwise the operation shall return false and fill the `SecurityException`.

Parameter exception: A `SecurityException` object.

Return: If an error occurs, this method shall return `false` and fill the `SecurityException`. otherwise it shall return the `IdentityStatusToken`.

8.3.2.11.11 Operation: `set_permissions_credential_and_token`

Associates the `PermissionsCredentialToken` (see 8.4.2.2) returned by the `AccessControl` plugin operation `get_permissions_credential_token` with the local `DomainParticipant` identified by the `IdentityHandle`.

This operation shall be called by the middleware after calling `validate_local_identity` and prior to any calls to `validate_remote_identity`.

Parameter handle: The handle used to locally identify the `DomainParticipant` whose `PermissionsCredential` is being supplied. The handle must have been returned by a successful call to *validate_local_identity*, otherwise the operation shall return false and fill the `SecurityException`.

Parameter permissions_credential_token: The `PermissionsCredentialToken` associated with the `DomainParticipant` identified by the `IdentityHandle`. The *permissions_credential_token* must have been returned by a successful call to `get_permissions_credential_token`, on the `AccessControl` plugin. Otherwise the operation shall return false and fill the `SecurityException`.

Parameter exception: A `SecurityException` object.

Return: If an error occurs, this method shall return `false`, otherwise it shall return `true`.

8.3.2.11.12 Operation: set_listener

Sets the `AuthenticationListener` that the `Authentication` plugin will use to notify the DDS middleware infrastructure of events relevant to the Authentication of DDS Participants.

If an error occurs, this method shall return `false` and fill the `SecurityException`.

Parameter listener: An `AuthenticationListener` object to be attached to the `Authentication` object. If this argument is `nil`, it indicates that there shall be no listener.

Parameter exception: A `SecurityException` object, which provides details in case the operation returns `false`.

8.3.2.11.13 Operation: return_identity_token

Returns the `IdentityToken` object to the plugin so it can be disposed of.

Parameter token: An `IdentityToken` issued by the plugin on a prior call to `get_identity_token`.

Parameter exception: A `SecurityException` object, which provides details in the case this operation returns `false`.

8.3.2.11.14 Operation: return_identity_status_token

Returns the `IdentityStatusToken` object to the plugin so it can be disposed of.

Parameter token: An `IdentityStatusToken` issued by the plugin on a prior call to `get_identity_status_token`.

Parameter exception: A `SecurityException` object, which provides details in the case this operation returns `false`.

8.3.2.11.15 Operation: return_authenticated_peer_credential_token

Returns the `AuthenticatedPeerCredentialToken` object to the plugin so it can be disposed of.

Parameter peer_credential_token: An `AuthenticatedPeerCredentialToken` issued by the plugin on a prior call to `get_authenticated_peer_credential_token`.

Parameter exception: A `SecurityException` object, which provides details in the case this operation returns `false`.

8.3.2.11.16 Operation: return_handshake_handle

Returns the `HandshakeHandle` object to the plugin so it can be disposed of.

Parameter handshake_handle: A `HandshakeHandle` issued by the plugin on a prior call to *begin_handshake_request* or *begin_handshake_reply*.

Parameter exception: A `SecurityException` object, which provides details in the case this operation returns `false`.

8.3.2.11.17 Operation: return_identity_handle

Returns the `IdentityHandle` object to the plugin so it can be disposed of.

Parameter identity_handle: An `IdentityHandle` issued by the plugin on a prior call to *validate_local_identity* or *validate_remote_identity*.

Parameter exception: A `SecurityException` object, which provides details in the case this operation returns `false`.

8.3.2.11.18 Operation: return_sharedsecret_handle

Returns the `SharedSecretHandle` object to the plugin so it can be disposed of.

Parameter sharedsecret_handle: An `IdentityHandle` issued by the plugin on a prior call to *get_shared_secret*.

Parameter exception: A `SecurityException` object, which provides details in the case this operation returns `false`.

8.3.2.12 AuthenticationListener

The `AuthenticationListener` provides the means for notifying the DDS middleware infrastructure of events relevant to the authentication of DDS `DomainParticipant` entities. For example, identity certificates can expire; in this situation, the `AuthenticationPlugin` shall call the `AuthenticationListener` to notify the DDS implementation that the identity of a specific `DomainParticipant` is being revoked.

Table 25 – Authentication listener class

AuthenticationListener		
No Attributes		
Operations		
`on_revoke_identity`		`Boolean`
	`plugin`	`Authentication`
	`handle`	`IdentityHandle`
	`out: exception`	`SecurityException`
`on_status_changed`		`void`
	`plugin`	`Authentication`
	`handle`	`IdentityHandle`
	`status_kind`	`AuthStatusKind`
	`out: exception`	`SecurityException`

8.3.2.12.1 Enumeration: AuthStatusKind

The `AuthStatusKind` enumerates the kind of changes to the status of the `Authentication` plugin or underlying Identity that are notified via the `AuthenticationListener` operation `on_status_changed`. The possible values are described in the table below:

Table 26 – Description of the AuthStatusKind values

Value	Meaning
IDENTITY_STATUS	Indicates a change to an identity status.
	Identity Status changes are represented externally to the `Authentication` plugin with an `IdentityStatusToken` that can be retrieved via the operation `get_identity_status_token` on the `Authentication` interface.
	The changed `IdentityStatusToken` shall be propagated by the DDS implementation to the other `DomainParticipants` using the ***DCPSParticipantsSecure*** builtin `Topic`.

8.3.2.12.2 Operation: `on_revoke_identity`

Revokes the identity of the participant identified by the `IdentityHandle`. The corresponding `IdentityHandle` becomes invalid. As a result of this, the DDS middleware shall terminate any communications with the `DomainParticipant` associated with that handle.

If an error occurs, this method shall return `false`.

Parameter plugin: An `Authentication` plugin object that has this listener allocated.

Parameter handle: An `IdentityHandle` object that corresponds to the Identity of a DDS Participant whose identity is being revoked.

8.3.2.12.3 Operation: `on_status_changed`

Informs the `DomainParticipant` that a status associated with the Authentication plugin or an Identity managed by the plugin has changed.

Depending on the kind of status the DDS implementation may need to take specific actions to retrieve information on the changed status and propagate it to other `DomainParticipant` entities. The actions that shall be taken for each kind of status are described in clause 8.3.2.12.1 .

8.4   Access Control Plugin

The Access Control Plugin API defines the types and operations necessary to support an access control mechanism for DDS `DomainParticipants`.

8.4.1 Background (Non-Normative)

Once a `DomainParticipant` is authenticated, its permissions need to be validated and enforced. Permissions or access rights are often described using an access control matrix where the rows are subjects (i.e., users), the columns are objects (i.e., resources), and a cell defines the access rights that a given subject has over a resource. Typical implementations provide either a column-centric view (i.e., access control lists) or a row-centric view (i.e., a set of capabilities stored with each subject). With the proposed `AccessControl` SPI, both approaches can be supported.

Before we can describe the access control plugin SPI, we need to define the permissions that can be attached to a `DomainParticipant`. Every DDS application uses a `DomainParticipant` to access or produce information on a `Domain`; hence the `DomainParticipant` has to be allowed to run in a certain `Domain`. Moreover, a `DomainParticipant` is responsible for creating `DataReaders` and `DataWriters` that communicate over a certain `Topic`. Hence, a `DomainParticipant` has to have the permissions needed to create a `Topic`, to publish through its `DataWriters` certain `Topics`, and to subscribe via its `DataReaders` to certain `Topics`. There is a very strong relationship between the `AccessControl` plugin and the `Cryptographic` plugin because encryption keys need to be generated for `DataWriters` based on the `DomainParticipant`'s permissions.

8.4.2 AccessControl Plugin Model

The `AccessControl` plugin model is presented in the figure below.
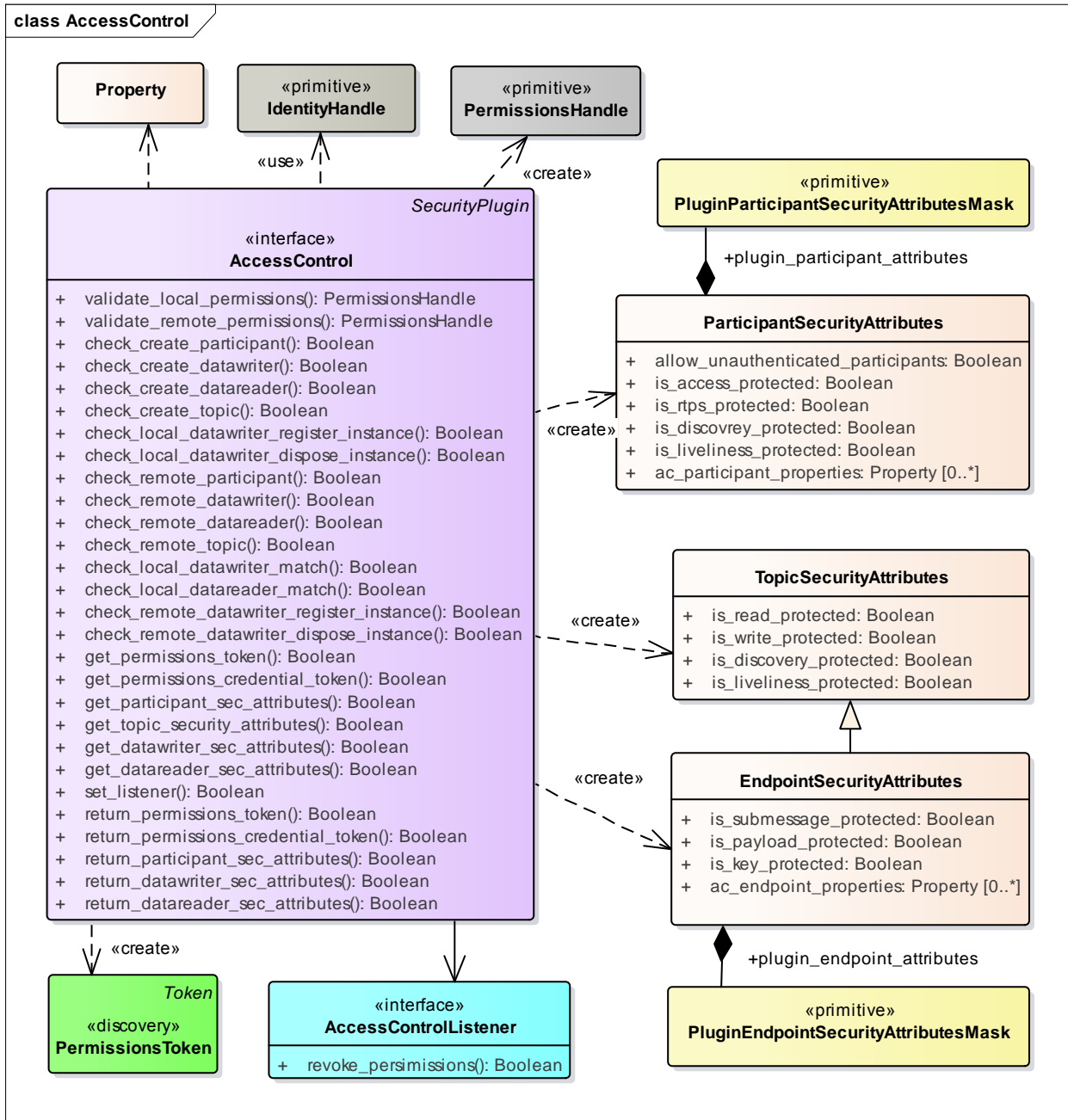
8.4.2.1 PermissionsToken

A `PermissionsToken` contains summary information on the permissions for a `DomainParticipant` in a manner that can be externalized and propagated over DDS discovery. The specific content of the `PermissionsToken` shall be defined by each `AccessControlPlugin` specialization. The intent is to provide only summary information on the permissions or derived information such as a hash.

8.4.2.2 PermissionsCredentialToken

A `PermissionsCredentialToken` encodes the permissions and access information for a `DomainParticipant` in a manner that can be externalized and sent over the network. The `PermissionsCredential` is used by the `AccessControl` plugin to verify the permissions of a peer `DomainParticipant` and perform all the access-control decisions related to that peer `DomainParticipant`, including determining whether it can join a domain, match specific local `DataWriters` or `DataReaders`, etc.

The `PermissionsCredentialToken` is intended for dissemination during the authentication handshake. The specific content of the `PermissionsCredentialToken` shall be defined by each `AccessControl` plugin specialization and it may not be used by some `AccessControl` plugin specializations.

8.4.2.3 PermissionsHandle

A `PermissionsHandle` is an opaque local reference to internal state within the `AccessControl` plugin. It is understood only by the `AccessControl` plugin and characterizes the permissions associated with a specific `DomainParticipant`. This object is returned by the `AccessControl` plugin as part of the validation of the permissions of a `DomainParticipant` and is used whenever a client of the `AccessControl` plugin needs to refer to the permissions of a previously validated `DomainParticipant`.

8.4.2.4 ParticipantSecurityAttributes

The `ParticipantSecurityAttributes` describe how the middleware should protect the `DomainParticipant`. This is a structured type whose members are described in the table below:

Table 27 – Description of the ParticipantSecurityAttributes

Member	Type	Meaning
allow_unauthenticated_participants	Boolean	Indicates whether the matching of the DomainParticipant with a remote DomainParticipant requires successful authentication. If *allow_unauthenticated_participants* is TRUE, the DomainParticipant shall allow matching other DomainParticipants—even if the remote DomainParticipant cannot authenticate—as long as there is no already a valid authentication with the same DomainParticipant's GUID. Additionally, a DomainParticipant that later authenticates would kick out the unauthenticated DomainParticipant if it has the same GUID. If *allow_unauthenticated_participants* is FALSE, the DomainParticipant shall enforce the authentication of remote DomainParticipants and disallow matching those that cannot be successfully authenticated.
is_access_protected	Boolean	Indicates whether the matching of the DomainParticipant with a remote DomainParticipant requires authorization by the AccessControl plugin.

		If *is_access_protected* is TRUE, then the DDS middleware shall call and `get_authenticated_peer_credential_token`, `validate_remote_permissions`, and `check_remote_participant` operations on the matched and authenticated remote DomainParticipant. Any failure in these calls will result in failing to authorize the remote participant, which shall be removed by the local participant.
		If *is_access_protected* is FALSE, then the DDS middleware shall call `get_authenticated_peer_credential_token` and `validate_remote_permissions` operations on the matched and authenticated remote DomainParticipant. However, a HandleNIL return from these operations will not prevent authorization..
is_rtps_prote cted	Boolean	Indicates whether the whole RTPS Message needs to be transformed by the CryptoTransform operation encode_rtps_message. If *is_rtps_protected* is TRUE then: (1) *allow_unauthenticated_participants* must be FALSE. (2) The DDS middleware shall call the operations on the CryptoKeyFactory for the DomainParticipant. (3) The DDS middleware shall call the operations on the CryptoKeyExchange for matched DomainParticipants that have been authenticated. (4) The RTPS messages sent by the DomainParticipant to matched DomainParticipants that have been authenticated shall be transformed using the CryptoTransform operation encode_rtps_message and the messages received from the matched authenticated DomainParticipants shall be transformed using the CryptoTransform operation decode_rtps_message, except for RTPS messages that contain submessages for any of the following builtin topics *"DCPSParticipants"*, *"DCPSParticipantStatelessMessage"*, or *"DCPSParticipantVolatileMessageSecure"*. These RTPS messages shall not be transformed by `encode_rtps_message`/`decode_rtps_message` operations. (5) RTPS messages that contain submessages for the builtin topics *"DCPSParticipants"*, *"DCPSParticipantStatelessMessage"*, or *"DCPSParticipantVolatileMessageSecure"* cannot contain submessages for any other builtin topic or application-defined topic. If *is_rtps_protected* is FALSE then the above actions shall not be taken.
is_discovery_ protected	Boolean	Indicates the DDS middleware shall call the operations on the `CryptoKeyFactory`, `CryptoKeyExchange`, and `CryptoTransform` for the *DCPSPublicationsSecure* and

		DCPSSubscriptionsSecure entities:
		If *is_discovery_protected* is TRUE then the `CryptoKeyFactory`, `CryptoKeyExchange` operations shall be called for the *DCPSPublicationsSecure* and *DCPSSubscriptionsSecure* entities to create the associated cryptographic material and send it to the matched entities.
		If *is_discovery_protected* is FALSE then the `CryptoKeyFactory`, `CryptoKeyExchange` and `CryptoTransform` operations will not be called.
		If *is_discovery_protected* is TRUE, the submessages sent by the *DCPSPublicationsSecure* and *DCPSSubscriptionsSecure* `DataWriters` shall be transformed using the `CryptoTransform` operation `encode_datawriter_submessage` and the messages received from the matched `DataReaders` shall be transformed using the `CryptoTransform` operation `decode_datareader_submessage`.
		If *is_discovery_protected* is TRUE, the submessages sent by the *DCPSPublicationsSecure* and *DCPSSubscriptionsSecure* DataReaders shall be transformed using the `CryptoTransform` operation `encode_datareader_submessage` and the messages received from the matched `DataWriters` shall be transformed using the `CryptoTransform` operation `decode_datawriter_submessage`.
		Independent of the setting of *is_discovery_protected*, the `CryptoTransform` operations `encode_serialized_payload` and `decode_serialized_payload` shall never be called for the *DCPSPublicationsSecure* and *DCPSSubscriptionsSecure* entities.

is_liveliness_ protected	Boolean	Indicates the DDS middleware shall call the operations on the `CryptoKeyFactory`, `CryptoKeyExchange`, and `CryptoTransform` for the **BuiltinParticipantMessageSecure** entities:
		If **is_liveliness_protected** is TRUE then the `CryptoKeyFactory`, `CryptoKeyExchange` operations shall be called for the **BuiltinParticipantMessageSecure** entities to create the associated cryptographic material and send it to the matched entities.
		If **is_liveliness_protected** is FALSE then the `CryptoKeyFactory`, **CryptoKeyExchange** and `CryptoTransform` operations will not be called.
		If **is_liveliness_protected** is TRUE, the submessages sent by the **BuiltinParticipantMessageSecure** `DataWriter` shall be transformed using the `CryptoTransform` operation `encode_datawriter_submessage` and the messages received from the matched `DataReaders` shall be transformed using the `CryptoTransform` operation `decode_datareader_submessage`.
		If **is_liveliness_protected** is TRUE, the submessages sent by the **BuiltinParticipantMessageSecure** `DataReader` shall be transformed using the `CryptoTransform` operation `encode_datareader_submessage` and the messages received from the matched `DataWriters` shall be transformed using the `CryptoTransform` operation `decode_datawriter_submessage`.
		Independent of the setting of **is_liveliness_protected**, the `CryptoTransform` operations `encode_serialized_payload` and `decode_serialized_payload` shall never be called for the **BuiltinParticipantMessageSecure** entities.
plugin_partici pant_attribut es	PluginP articipa ntSecuri tyAttrib utesMas k	This field is a holder for plugin-specific information that is propagated via discovery as part of the ParticipantSecurityInfo (see 7.2.7). The definition for the builtin plugins can be found in clause 9.4.2.4.
ac_participan t_properties	Propert ySeq	Additional properties to add to the **participant_properties** parameter passed to the `CryptoKeyFactory` operation `register_local_participant`. See 8.5.1.7.1. The returned **ac_participant_properties** and their interpretation shall be specified by each plugin implementation.

8.4.2.5 Definition of the ParticipantSecurityAttributesMask

The `ParticipantSecurityAttributesMask` is used to encode the value of the `ParticipantSecurityAttributes` in a compact way such that it can be included in the `ParticipantSecurityInfo`, see 7.2.8.

The mapping of the `ParticipantSecurityAttributes` to `ParticipantSecurityAttributesMask` shall be as follows:

Table 28 – Mapping of fields ParticipantSecurityAttributes to bits in ParticipantSecurityAttributesMask

Field in ParticipantSecurityAttributes	Corresponding bit in the ParticipantSecurityAttributesMask
allow_unauthenticated_participants	No mapping, this attribute is not checked remotely.
is_access_protected	No mapping, this attribute is not checked remotely.
is_rtps_protected	#define PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_RTPS_PROTECTED (0x00000001 << 0)
is_discovery_protected	#define PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_DISCOVERY_PROTECTED (0x00000001 << 1)|
is_liveliness_protected	#define PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_LIVELINESS_PROTECTED (0x00000001 << 2)
is_valid	#define PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_VALID (0x00000001 << 31)|

8.4.2.6 TopicSecurityAttributes

The `TopicSecurityAttributes` describe how the middleware shall protect the Entity. This is a structured type, whose members are described in the table below:

Table 29 – Description of the TopicSecurityAttributes

Member	Type	Meaning
is_read_protected	Boolean	Indicates if read access to the `Topic` is protected. If *is_read_protected* is FALSE, then local `DataReader` creation and remote `DataReader` matching can proceed without further access-control mechanisms imposed. Otherwise, they shall be checked using the `AccessControl` operations.
is_write_protected	Boolean	Indicates if read access to the `Topic` is protected. If *is_write_protected* is FALSE, then local `DataWriter` creation and remote `DataWriter` matching can proceed without further access-control mechanisms imposed. Otherwise, they shall be checked using the `AccessControl` operations.
is_discovery_protected	Boolean	Indicates the discovery information for the entity shall be sent using a secure builtin discovery topics or the regular builtin discovery topics: If *is_discovery_protected* is TRUE then discovery information for that entity shall be sent using the ***SEDPbuiltinPublicationsSecureWriter SEDPbuiltinSubscriptionsSecureWriter***. If *is_discovery_protected* is FALSE then discovery information for that entity shall be sent using the ***SEDPbuiltinPublicationsWriter*** or ***SEDPbuiltinSubscriptionsWriter***.
is_liveliness_protected	Boolean	The value of this attribute matters only if the `DataWriter LivelinessQos` policy is AUTOMATIC_LIVELINESS_QOS or MANUAL_BY_PARTICIPANT_LIVELINESS_QOS. In this case it indicates whether the liveliness information for the entity shall be sent using the ***BuiltinParticipantMessage*** or the ***BuiltinParticipantMessageSecure*** builtin `Topic`. If *is_liveliness_protected* is TRUE then the liveliness heartbeats are sent using the ***BuiltinParticipantMessageSecure*** builtin `Topic`. Otherwise they are sent using the ***BuiltinParticipantMessage*** builtin `Topic`.

8.4.2.7 EndpointSecurityAttributes

The `EndpointSecurityAttributes` describe how the middleware shall protect the Entity. This is a structured type, derived from TopicSecurityAttributes, whose additional members are described in the table below:

Table 30 – Description of the EndpointSecurityAttributes

Member	Type	Meaning
is_submessage_protected	Boolean	Indicates the DDS middleware shall call the operations on the CryptoKeyFactory, CryptoKeyExchange, and CryptoTransform for the entity: If *is_submessage_protected* is TRUE, then the CryptoKeyFactory, CryptoKeyExchange operations shall be called for that entity to create the associated cryptographic material and send it to the matched entities. If *is_submessage_protected* is FALSE, then the CryptoKeyFactory, CryptoKeyExchange and CryptoTransform operations are called only if is_payload_protected is TRUE. If *is_submessage_protected* is TRUE and the entity is a DataWriter, the submessages sent by the DataWriter shall be transformed using the CryptoTransform operation encode_datawriter_submessage and the messages received from the matched DataReaders shall be transformed using the CryptoTransform operation decode_datareader_submessage. If *is_submessage_protected* is TRUE and the entity is a DataReader, the submessages sent by the DataReader shall be transformed using the CryptoTransform operation encode_datareader_submessage and the messages received from the matched DataWriters shall be transformed using the CryptoTransform operation decode_datawriter_submessage.

is_payload_protected	Boolean	Indicates the DDS middleware shall call the operations on the CryptoKeyFactory, CryptoKeyExchange, and CryptoTransform for the entity.
		If *is_payload_protected* is TRUE, then the CryptoKeyFactory, CryptoKeyExchange operations shall be called for that entity to create the associated cryptographic material and send it to the matched entities.
		If *is_payload_protected* is FALSE, then the CryptoKeyFactory, CryptoKeyExchange and CryptoTransform operations are called only if is_payload_protected is TRUE.
		If *is_ payload_protected* is TRUE and the entity is a DataWriter, the serialized data sent by the DataWriter shall be transformed by calling encode_serialized_payload.
		If *is_ payload_protected* is TRUE and the entity is a DataReader, the serialized data received by the DataReader shall be transformed by calling decode_serialized_payload
is_key_protected		Indicates that the content of the Instance Key is sensitive.
		If *is_key_protected* is TRUE, then the DDS middleware shall compute the KeyHash for the Instance Key as described in section 7.3.4.
		If *is_key_protected* is FALSE, then the DDS middleware should the compute the KeyHash for the Instance Key as described in clause 9.6.3.3 of the DDS-RTPS

		specification [2].
plugin_endpoint_attribu tes	PluginEndpointSpecificAttributesM ask	This field is a holder for plugin-specific information that is propagated via discovery as part of the EndpointSecurityInfo (see 7.2.8). The definition for the builtin plugins can be found in 0
ac_endpoint_properties	PropertySeq	Additional properties to add to the *datawriter_properties* or *datareader_properties* passed to the `CryptoKeyFactory` operations `register_local_datawrit er` and `register_local_dataread er`. The returned *ac_endpoint_properties* and their interpretation shall be specified by each plugin implementation.

8.4.2.8   Definition of the EndpointSecurityAttributesMask

The `EndpointSecurityAttributesMask` is used to encode the value of the `EndpointSecurityAttributes` in a compact way such that it can be included in the `EndpointSecurityInfo`, see 7.2.8

The mapping of the `EndpointSecurityAttributes` to `EndpointSecurityAttributesMask` shall be as defined in the table below:

Table 31 – Mapping of fields EndpointSecurityAttributes to bits in EndpointSecurityAttributesMask

Field in EndpointSecurityAtt ributes	Corresponding bit in the `EndpointSecurityAttributesMask`
is_read_protected	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_READ_PROTECTED (0x00000001 << 0)
is_write_protected	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_WRITE_PROTECTED (0x00000001 << 1)
is_discovery_protect ed	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_DISCOVERY_PROTECTED (0x00000001 << 2)

is_submessage_protected	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_SUBMESSAGE_PROTECTED (0x00000001 << 3)
is_payload_protected	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_PAYLOAD_PROTECTED (0x00000001 << 4)
is_key_protected	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_KEY_PROTECTED (0x00000001 << 5)
is_liveliness_protected	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_LIVELINESS_PROTECTED (0x00000001 << 6)
is_valid	#define ENDPOINT_SECURITY_ATTRIBUTES_FLAG_IS_VALID (0x00000001 << 31)

8.4.2.9 AccessControl interface

Table 32 – AccessControl Interface

AccessControl		
No Attributes		
Operations		
validate_local_permissions		PermissionsHandle
	auth_plugin	AuthenticationPlugin
	identity	IdentityHandle
	domain_id	DomainId_t
	participant_qos	DomainParticipantQos
	out: exception	SecurityException
validate_remote_permissions		PermissionsHandle
	auth_plugin	AuthenticationPlugin
	local_identity_handle	IdentityHandle
	remote_identity_handle	IdentityHandle
	remote_permissions_token	PermissionsToken
	remote_credential_	AuthenticatedPeerCredenti

	token	alToken
	out: exception	SecurityException
check_create_partic ipant		Boolean
	permissions_handle	PermissionsHandle
	domain_id	DomainId_t
	qos	DomainParticipantQoS
	out: exception	SecurityException

check_create_datawriter		Boolean
	permissions_handle	PermissionsHandle
	domain_id	DomainId_t
	topic_name	String
	qos	DataWriterQoS
	partition	PartitionQosPolicy
	data_tag	DataTag
	out: exception	SecurityException
check_create_datareader		Boolean
	permissions_handle	PermissionsHandle
	domain_id	DomainId_t
	topic_name	String
	qos	DataReaderQoS
	partition	PartitionQosPolicy
	data_tag	DataTag
	out: exception	SecurityException
check_create_topic		Boolean
	permissions_handle	PermissionsHandle
	domain_id	DomainId_t
	topic_name	String
	qos	TopicQoS
	out: exception	SecurityException
check_local_datawriter_register_instance		Boolean
	permissions_handle	PermissionsHandle
	writer	DataWriter
	key	DynamicData
	out: exception	SecurityException

		Boolean
check_local_datawri ter_dispose_instanc e	permissions_handle	PermissionsHandle
	writer	DataWriter
	key	DynamicData
	out: exception	SecurityException
check_remote_partic ipant		Boolean
	permissions_handle	PermissionsHandle
	domain_id	DomainId_t
	participant_data	ParticipantBuiltinTopicDa taSecure
	out: exception	SecurityException
check_remote_datawr iter		Boolean
	permissions_handle	PermissionsHandle
	domain_id	DomainId_t
	publication_data	PublicationBuiltinTopicDa taSecure
	out: exception	SecurityException
check_remote_datare ader		Boolean
	permissions_handle	PermissionsHandle
	domain_id	DomainId_t
	subscription_data	SubscriptionBuiltinTopicD ataSecure
	out: relay_only	Boolean
	out: exception	SecurityException
check_remote_topic		Boolean
	permissions_handle	PermissionsHandle
	DomainId_t	domain_id
	topic_data	TopicBuiltinTopicData
	out: exception	SecurityException

check_local_datawri ter_match		Boolean
	writer_permissions _handle	PermissionsHandle
	reader_permissions _handle	PermissionsHandle
	publication_data	PublicationBuiltInTopicDa taSecure
	subscription_data	PublicationBuiltinTopicDa taSecure
	out: exception	SecurityException
check_local_datarea der_match		Boolean
	reader_permissions _handle	PermissionsHandle
	writer_permissions _handle	PermissionsHandle
	subscriber_partiti on	PartitionQosPolicy
	publication_data	PublicationBuiltInTopicDa taSecure
	subscription_data	PublicationBuiltinTopicDa taSecure
	out: exception	SecurityException
check_remote_datawr iter_register_insta nce		Boolean
	permissions_handle	PermissionsHandle
	reader	DataReader
	publication_handle	InstanceHandle_t
	key	DynamicData
	instance_handle	InstanceHandle_t
	out: exception	SecurityException

check_remote_datawr iter_dispose_instan ce		Boolean
	permissions_handle	PermissionsHandle
	reader	DataReader
	publication_handle	InstanceHandle_t
	key	DynamicData
	out: exception	SecurityException
get_permissions_tok en		Boolean
	out: permissions_token	PermissionsToken
	handle	PermissionsHandle
	out: exception	SecurityException
get_permissions_cre dential_token		Boolean
	out: permissions_creden tial_token	PermissionsCredentialToke n
	handle	PermissionsHandle
	out: exception	SecurityException
set_listener		Boolean
	listener	AccessControlListener
	out: exception	SecurityException
return_permissions_ token		Boolean
	token	PermissionsToken
	out: exception	SecurityException
return_permissions_ credential_token		Boolean
	permissions_creden tial_token	PermissionsCredentialToke n
	out: exception	SecurityException
get_participant_sec _attributes		Boolean
	permissions_handle	PermissionsHandle
	out: attributes	ParticipantSecurityAttrib utes
	out: exception	SecurityException

get_topic_sec_attri butes		Boolean
	permissions_handle	PermissionsHandle
	topic_name	string
	out: attributes	EndpointSecurityAttribute s
	out: exception	SecurityException
get_datawriter_sec_ attributes		Boolean
	permissions_handle	PermissionsHandle
	topic_name	string
	partition	PartitionQosPolicy
	data_tag	DataTagQosPolicy
	out: attributes	EndpointSecurityAttribute s
	out: exception	SecurityException
get_datareader_sec_ attributes		Boolean
	permissions_handle	PermissionsHandle
	topic_name	string
	partition	PartitionQosPolicy
	data_tag	DataTagQosPolicy
	out: attributes	EndpointSecurityAttribute s
	out: exception	SecurityException
return_participant_ sec_attributes		Boolean
	attributes	ParticipantSecurityAttrib utes
	out: exception	SecurityException
return_datawriter_s ec_attributes		Boolean
	attributes	EndpointSecurityAttribute s
	out: exception	SecurityException

return_datareader_s ec_attributes		Boolean
	attributes	EndpointSecurityAttribute s
	out: exception	SecurityException

8.4.2.9.1 Operation: `validate_local_permissions`

Validates the permissions of the local `DomainParticipant`. The operation returns a `PermissionsHandle` object, if successful. The `PermissionsHandle` can be used to locally identify the permissions of the local `DomainParticipant` to the `AccessControl` plugin.

This operation shall be called before the `DomainParticipant` is enabled. It shall be called either by the implementation of `DomainParticipantFactory create_domain_participant` or `DomainParticipant enable` [1].

If an error occurs, this method shall return `HandleNIL`.

Parameter auth_plugin: The `Authentication` plugin, which validated the identity of the local `DomainParticipant`. If this argument is `nil`, the operation shall return `HandleNIL`.

Parameter identity: The `IdentityHandle` returned by the authentication plugin from a successful call to `validate_local_identity`.

Parameter domain_id: The DDS Domain Id of the `DomainParticipant`.

Parameter participant_qos: The `DomainParticipantQos` of the `DomainParticipant`.

Parameter exception: A SecurityException object, which provides details, in case this operation returns `HandleNIL`.

8.4.2.9.2 Operation: `validate_remote_permissions`

Validates the permissions of the previously authenticated remote `DomainParticipant`, given the `PermissionsToken` object received via DDS discovery and the `PermissionsCredentialToken` obtained as part of the authentication process. The operation returns a `PermissionsHandle` object, if successful.

If an error occurs, this method shall return `HandleNIL`.

Parameter auth_plugin: The `Authentication` plugin, which validated the identity of the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `HandleNIL`.

Parameter local_identity_handle: The `IdentityHandle` returned by the authentication plugin.

Parameter remote_identity_handle: The `IdentityHandle` returned by a successful call to the `validate_remote_identity` operation on the `Authentication` plugin.

Parameter remote_permissions_token: The `PermissionsToken` of the remote `DomainParticipant` received via DDS discovery inside the ***permissions_token*** member of the ***ParticipantBuiltinTopicData***. See 7.4.1.3.

Parameter remote_credential_token: The `AuthenticatedPeerCredentialToken` of the remote `DomainParticipant` returned by the operation `get_authenticated_peer_credential_token` on the `Authentication` plugin.

Parameter exception: A `SecurityException` object, which provides details, in case this operation returns `HandleNIL`.

8.4.2.9.3 Operation: `check_create_participant`

Enforces the permissions of the local `DomainParticipant`. When the local `DomainParticipant` is created, its permissions must allow it to join the DDS `Domain` specified by the ***domain_id***. Optionally the use of the specified value for the `DomainParticipantQoS` must also be allowed by its permissions. The operation returns a `Boolean` value.

This operation shall be called before the `DomainParticipant` is enabled. It shall be called either by the implementation of `DomainParticipantFactory create_domain_participant` or `DomainParticipant enable` [1].

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter domain_id: The domain id where the local `DomainParticipant` is about to be created. If this argument is `nil`, the operation shall return `false`.

Parameter qos: The QoS policies of the local `DomainParticipant`. If this argument is nil, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.4 Operation: `check_create_datawriter`

Enforces the permissions of the local `DomainParticipant`. When the local `DomainParticipant` creates a `DataWriter` for `topic_name` with the specified `DataWriterQos` associated with the `data_tag`, its permissions must allow this. The operation returns a `Boolean` object.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter domain_id: The `DomainId_t` of the local `DomainParticipant` to which the local `DataWriter` will belong.

Parameter topic_name: The topic name that the `DataWriter` is supposed to write. If this argument is `nil`, the operation shall return `false`.

Parameter qos: The QoS policies of the local `DataWriter`. If this argument is nil, the operation shall return `false`.

Parameter partition: The `PartitionQosPolicy` of the local `Publisher` to which the `DataWriter` will belong.

Parameter data_tag: The data tags that the local `DataWriter` is requesting to be associated with its data. This argument can be `nil` if it is not considered for access control.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.5   Operation: `check_create_datareader`

Enforces the permissions of the local `DomainParticipant`. When the local `DomainParticipant` creates a `DataReader` for a `Topic` for `topic_name` with the specified `DataReaderQos` qos associated with the `data_tag`, its permissions must allow this. The operation returns a `Boolean` value.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter domain_id: The `DomainId_t` of the local `DomainParticipant` to which the local `DataReader` will belong.

Parameter topic_name: The topic name that the `DataReader` is supposed to read. If this argument is `nil`, the operation shall return `false`.

Parameter qos: The QoS policies of the local `DataReader`. If this argument is nil, the operation shall return `false`.

Parameter partition: The `PartitionQosPolicy` of the local `Subscriber` to which the `DataReader` will belong.

Parameter data_tag: The data tags that the local `DataReader` is requesting read access to. This argument can be `nil` if it is not considered for access control.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.6   Operation: `check_create_topic`

Enforces the permissions of the local `DomainParticipant`. When an entity of the local `DomainParticipant` creates a `Topic` with `topic_name` and `TopicQos` qos its permissions must allow this. The operation returns a `Boolean` value.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter domain_id: The `DomainId_t` of the local `DomainParticipant` that creates the Topic.

Parameter topic_name: The topic name to be created. If this argument is `nil`, the operation shall return `false`.

Parameter qos: The QoS policies of the local `Topic`. If this argument is nil, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.7   Operation: `check_local_datawriter_register_instance`

Enforces the permissions of the local `DomainParticipant`. In case the access control requires a finer granularity at the instance level, this operation enforces the permissions of the local `DataWriter`. The `key` identifies the instance being registered and permissions are checked to determine if registration of the specified instance is allowed. The operation returns a `Boolean` value.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter writer: `DataWriter` object that registers the instance. If this argument is `nil`, the operation shall return `false`.

Parameter key: The `key` of the instance for which the registration permissions are being checked. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.8   Operation: `check_local_datawriter_dispose_instance`

Enforces the permissions of the local `DomainParticipant`. In case the access control requires a finer granularity at the instance level, this operation enforces the permissions of the local `DataWriter`. The `key` has to match the permissions for disposing an instance. The operation returns a `Boolean` object.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter writer: `DataWriter` object that registers the instance. If this argument is `nil`, the operation shall return `false`.

Parameter key: The `key` identifies the instance being registered and the permissions are checked to determine if disposal of the specified instance is allowed. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `nil`.

8.4.2.9.9 Operation: `check_remote_participant`

Enforces the permissions of the remote `DomainParticipant`. When the remote `DomainParticipant` is discovered, the domain_id and, optionally, the `DomainParticipantQoS` are checked to verify that joining that DDS Domain and using that QoS is allowed by its permissions. The operation returns a `Boolean` result.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter domain_id: The domain id where the remote `DomainParticipant` is about to be created. If this argument is `nil`, the operation shall return `false`.

Parameter participant_data: The `ParticipantBuiltInTopicDataSecure` object associated with the remote `DomainParticipant`. If this argument is nil, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `nil`.

8.4.2.9.10 Operation: `check_remote_datawriter`

Enforces the permissions of a remote `DomainParticipant`.

This operation shall be called by a `DomainParticipant` prior to matching a local `DataReader` belonging to that `DomainParticipant` with a `DataWriter` belonging to a different (peer) `DomainParticipant`.

This operation shall also be called whenever a `DomainParticipant` detects a QoS change for a `DataWriter` belonging to a different (peer) `DomainParticipant` that is matched with a local `DataReader`.

This operation verifies that the peer `DomainParticipant` has the permissions necessary to publish data on the DDS `Topic` with name *topic_name* using the `DataWriterQoS` that appears in *publication_data*. The operation returns a `Boolean` value.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter domain_id: The domain id of the `DomainParticipant` to which the remote `DataWriter` belongs.

Parameter publication_data: The `PublicationBuiltInTopicDataSecure` object associated with the remote `DataWriter`. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.11 Operation: `check_remote_datareader`

Enforces the permissions of a remote `DomainParticipant`.

This operation shall be called by a `DomainParticipant` prior to matching a local `DataWriter` belonging to that `DomainParticipant` with a `DataReader` belonging to a different (peer) `DomainParticipant`.

This operation shall also be called whenever a `DomainParticipant` detects a QoS change for a `DataReader` belonging to a different (peer) `DomainParticipant` that is matched with a local `DataWriter`.

This operation verifies that the peer `DomainParticipant` has the permissions necessary to subscribe to data on the DDS `Topic` with name *topic_name* using the `DataReaderQoS` that appears in *subscription_data*. The operation returns a `Boolean` value and also sets the *relay_only* output parameter.

If the operation returns `true`, the DDS middleware shall allow the local `DataWriter` to match with the remote `DataReader`, if it returns `false`, it shall not allow it.

If the operation returns `true`, the *relay_only* parameter shall be remembered by the DDS middleware and passed to the `register_matched_remote_datareader` operation on the `CryptoKeyFactory`.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter domain_id: The domain id of the `DomainParticipant` to which the remote `DataReader` belongs.

Parameter subscription_data: The `SubscriptionBuiltInTopicDataSecure` object associated with the remote `DataReader`. If this argument is `nil`, the operation shall return `false`.

Parameter (out) relay_only: Boolean indicating whether the permissions of the remote `DataReader` are restricted to relaying the information (understanding sequence numbers and other `SubmessageHeader` information) but not decoding the data itself. This parameter is only meaningful if the operation returns `true`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.12 Operation: `check_remote_topic`

Enforces the permissions of the remote `DomainParticipant`. When the remote `DomainParticipant` creates a certain topic, the *topic_name* and optionally the `TopicQoS` extracted from the *topic_data* are verified to ensure the remote `DomainParticipant` permissions allow it to create the DDS `Topic` with the specified QoS. The operation returns a `Boolean` value.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter topic_data: The `TopicBuiltInTopicData` object associated with the `Topic`. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.13 Operation: `check_local_datawriter_match`

Provides the means for the `AccessControl` plugin to enforce access control rules that are based on the `DataTag` associated with `DataWriter` and a matching `DataReader`.

The operation shall be called for any local `DataWriter` that matches a `DataReader`. The operation shall be called after the operation `check_local_datawriter` has been called on the local `DataWriter` and either `check_local_datareader` or `check_remote_datareader` has been called on the `DataReader`.

This operation shall also be called when a local `DataWriter`, matched with a `DataReader`, detects a change on the Qos of the `DataReader`.

The operation shall be called only if the aforementioned calls to `check_local_datawriter` and `check_local_datareader` or `check_remote_datareader` are returned successfully.

The operation returns a `Boolean` value. If an error occurs, this method shall return `false` and the `SecurityException` filled.

Parameter writer_permissions_handle: The `PermissionsHandle` object associated with the `DomainParticipant` that contains the local `DataWriter`. If this argument is `nil`, the operation shall return `false`.

Parameter reader_permissions_handle: The `PermissionsHandle` object associated with the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter publication_data: The `PublicationBuiltInTopicDataSecure` object associated with the local `DataWriter`. If this argument is `nil`, the operation shall return `false`.

Parameter subscription_data: The `SubscriptionBuiltInTopicDataSecure` object associated with the matched `DataReader`. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.14 Operation: `check_local_datareader_match`

Provides the means for the `AccessControl` plugin to enforce access control rules that are based on the `DataTag` associated with a `DataReader` and a matching `DataWriter`.

The operation shall be called for any local `DataReader` that matches a `DataWriter`. The operation shall be called after the operation `check_local_datareader` has been called on the local `DataReader` and either `check_local_datawriter` or `check_remote_datawriter` has been called on the `DataWriter`.

This operation shall also be called when a local `DataReader`, matched with a `DataWriter`, detects a change on the Qos of the `DataWriter`.

The operation shall be called only if the aforementioned calls to `check_local_datareader` and `check_local_datawriter` or `check_remote_datawriter` are returned successfully.

The operation returns a `Boolean` value. If an error occurs, this method shall return `false` and the `SecurityException` filled.

Parameter writer_permissions_handle: The `PermissionsHandle` object associated with the `DomainParticipant` that contains the local `DataReader`. If this argument is `nil`, the operation shall return `false`.

Parameter reader_permissions_handle: The `PermissionsHandle` object associated with the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter subscription_data: The `SubscriptionBuiltInTopicDataSecure` object associated with the local `DataReader`. If this argument is `nil`, the operation shall return `false`.

Parameter publication_data: The `PublicationBuiltInTopicDataSecure` object associated with the matched `DataWriter`. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.15 Operation: check_remote_datawriter_register_instance

Enforces the permissions of the remote `DomainParticipant`. In case the access control requires a finer granularity at the instance level, this operation enforces the permissions of the remote `DataWriter`. The `key` has to match the permissions for registering an instance. The operation returns a `Boolean` value.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter reader: The local `DataReader` object that is matched to the remote `DataWriter` that registered an instance.

Parameter publication handle: Handle that identifies the remote `DataWriter`.

Parameter key: The `key` of the instance that needs to match the permissions for registering an instance. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.16 Operation: check_remote_datawriter_dispose_instance

Enforces the permissions of the remote `DomainParticipant`. In case the access control requires a finer granularity at the instance level, this operation enforces the permissions of the remote `DataWriter`. The `key` has to match the permissions for disposing an instance. The operation returns a `Boolean` value.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the remote `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter reader: The local `DataReader` object that is matched to the Publication that disposed an instance.

Parameter publication handle: Handle that identifies the remote Publication.

Parameter key: The `key` of the instance that needs to match the permissions for disposing an instance. If this argument is `nil`, the operation shall return `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.17 Operation: `get_permissions_token`

Retrieves a `PermissionsToken` object. The `PermissionsToken` is propagated via DDS discovery to summarize the permissions of the `DomainParticipant` identified by the specified `PermissionsHandle`.

If an error occurs, this method shall return `false`.

Parameter permissions_token (out): The returned `PermissionsToken`.

Parameter handle: The handle used to locally identify the permissions of the DomainParticipant for which a `PermissionsToken` is desired. If this argument is `nil`, the operation shall return `nil`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.18 Operation: `get_permissions_credential_token`

Retrieves a `PermissionsCredentialToken` object that can be used to represent on the network the permissions of the `DomainParticipant` identified by the specified `PermissionsHandle`.

If an error occurs, this method shall return `false`.

Parameter permissions_credential_token (out): The returned `PermissionsCredentialToken`.

Parameter handle: The `PermissionsHandle` used to locally identify the permissions of the `DomainParticipant` for which a `PermissionsCredentialToken` is desired. If this argument is `nil`, the operation shall return `nil`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.19 Operation: `set_listener`

Sets the listener for the `AccessControl` plugin.

If an error occurs, this method shall return `false`.

Parameter listener: An `AccessControlListener` object to be attached to the `AccessControl` plugin. If this argument is `nil`, the operation returns `false`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.20 Operation: `return_permissions_token`

Returns the `PermissionsToken` to the plugin for disposal.

Parameter token: A `PermissionsToken` to be disposed of. It should correspond to the `PermissionsToken` returned by a prior call to `get_permissions_token` on the same plugin.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.21 Operation: `return_permissions_credential_token`

Returns the `PermissionsCredentialToken` to the plugin for disposal.

Parameter permissions_credential_token: A `PermissionsCredentialToken` to be disposed of. It should correspond to the `PermissionsCredentialToken` returned by a prior call to `get_permissions_credential_token` on the same plugin.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.22 Operation: get_participant_sec_attributes

Retrieves the `ParticipantSecurityAttributes`, which describe how the DDS middleware should enforce the security and integrity of the information produced and consumed via the DomainParticipant.

This operation shall be called by the DDS middleware as part of the creation or enabling of the DDS `DomainParticipant`.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter (out) attributes: The returned `ParticipantSecurityAttributes`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.23 Operation: get_topic_sec_attributes

Retrieves the `TopicSecurityAttributes`, which describes how the DDS middleware should enforce the security and integrity of the information related to the DDS `Topic`.

This operation shall be called by the DDS middleware as part of the creation or enabling of a DDS Topic. The operation shall be called before calling `check_create_topic`, `check_create_datawriter`, `check_create_datareader`, `check_remote_datawriter`, `check_remote_datareader`, `check_remote_datawriter_match`, or `check_remote_datareader_match`.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is nil, the operation shall return false.

Parameter topic_name: The name of the `Topic`. If this argument is nil, the operation shall return false.

Parameter (out) attributes: The returned `TopicSecurityAttributes`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns false.

8.4.2.9.24 Operation: get_datarwriter_sec_attributes

Retrieves the `EndpointSecurityAttributes`, which describes how the DDS middleware should enforce the security and integrity of the information related to the DDS `DataWriter` endpoint.

This operation shall be called by the DDS middleware as part of the creation or enabling of a DDS `DataWriter`. The operation shall be called after calling `check_create_datawriter`.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter topic_name: The name of the `Topic` associated with the `DataWriter`. If this argument is `nil`, the operation shall return `false`.

Parameter partition: The `PartitionQosPolicy` of the local `Publisher` to which the `DataWriter` belongs.

Parameter data_tag: The `DataTagQosPolicy` associated with the `DataWriter`. This argument can be `nil`.

Parameter (out) attributes: The returned `EndpointSecurityAttributes`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.25 Operation: get_datareader_sec_attributes

Retrieves the `EndpointSecurityAttributes`, which describes how the DDS middleware should enforce the security and integrity of the information related to the DDS `DataReader` endpoint.

This operation shall be called by the DDS middleware as part of the creation or enabling of a DDS `DataReader`. The operation shall be called after calling `check_create_datareader`.

If an error occurs, this method shall return `false`.

Parameter permissions_handle: The `PermissionsHandle` object associated with the local `DomainParticipant`. If this argument is `nil`, the operation shall return `false`.

Parameter topic_name: The name of the `Topic` associated with the `DataReader`. If this argument is `nil`, the operation shall return `false`.

Parameter partition: The `PartitionQosPolicy` of the local `Subscriber` to which the `DataReader` belongs.

Parameter data_tag: The data tag associated with the `DataReader`. This argument can be `nil`.

Parameter (out) attributes: The returned `EndpointSecurityAttributes`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.4.2.9.26 Operation: return_participant_sec_attributes:

Returns the `ParticipantSecurityAttributes` to the plugin for disposal.

Parameter attributes: The `ParticipantSecurityAttributes` to return.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns false.

8.4.2.9.27 Operation: return_datawriter_sec_attributes:

Returns the `EndpointSecurityAttributes` to the plugin for disposal.

Parameter attributes: The `EndpointSecurityAttributes` to return.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns false.

8.4.2.9.28 Operation: return_datareader_sec_attributes:

Returns the `EndpointSecurityAttributes` to the plugin for disposal.

Parameter attributes: The `EndpointSecurityAttributes` to return.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns false.

8.4.2.10 AccessControlListener interface

The purpose of the `AccessControlListener` is to be notified of all status changes for different identities. For example, permissions can change; hence, the `AccessControlListener` is notified and enforces the new permissions.

Table 33 – AccessControlListener interface

AccessControlListener		
No Attributes		
Operations		
`on_revoke_permissions`		`Boolean`
	`plugin`	`AccessControl`
	`handle`	`PermissionsHandle`

8.4.2.10.1 Operation: `on_revoke_permissions`

`DomainParticipants`' Permissions can be revoked/changed. This listener provides a callback for permission revocation/changes.

If an error occurs, this method shall return `false`.

Parameter plugin: The correspondent `AccessControl` object.

Parameter handle: A `PermissionsHandle` object that corresponds to the Permissions of a DDS Participant whose permissions are being revoked.

8.5 Cryptographic Plugin

The `Cryptographic` plugin defines the types and operations necessary to support encryption, digest, message authentication codes, and key exchange for DDS `DomainParticipants`, `DataWriters` and DDS `DataReaders`.

Users of DDS may have specific cryptographic libraries they use for encryption, as well as, specific requirements regarding the algorithms for digests, message authentication, and signing. In addition, applications may require having only some of those functions performed, or performed only for certain DDS `Topics` and not for others. Therefore, the plugin API has to be general enough to allow flexible configuration and deployment scenarios.

8.5.1 Cryptographic Plugin Model

The `Cryptographic` plugin model is presented in the figure below. It combines related cryptographic interfaces for key creation, key exchange, encryption, message authentication, hashing, and signature.
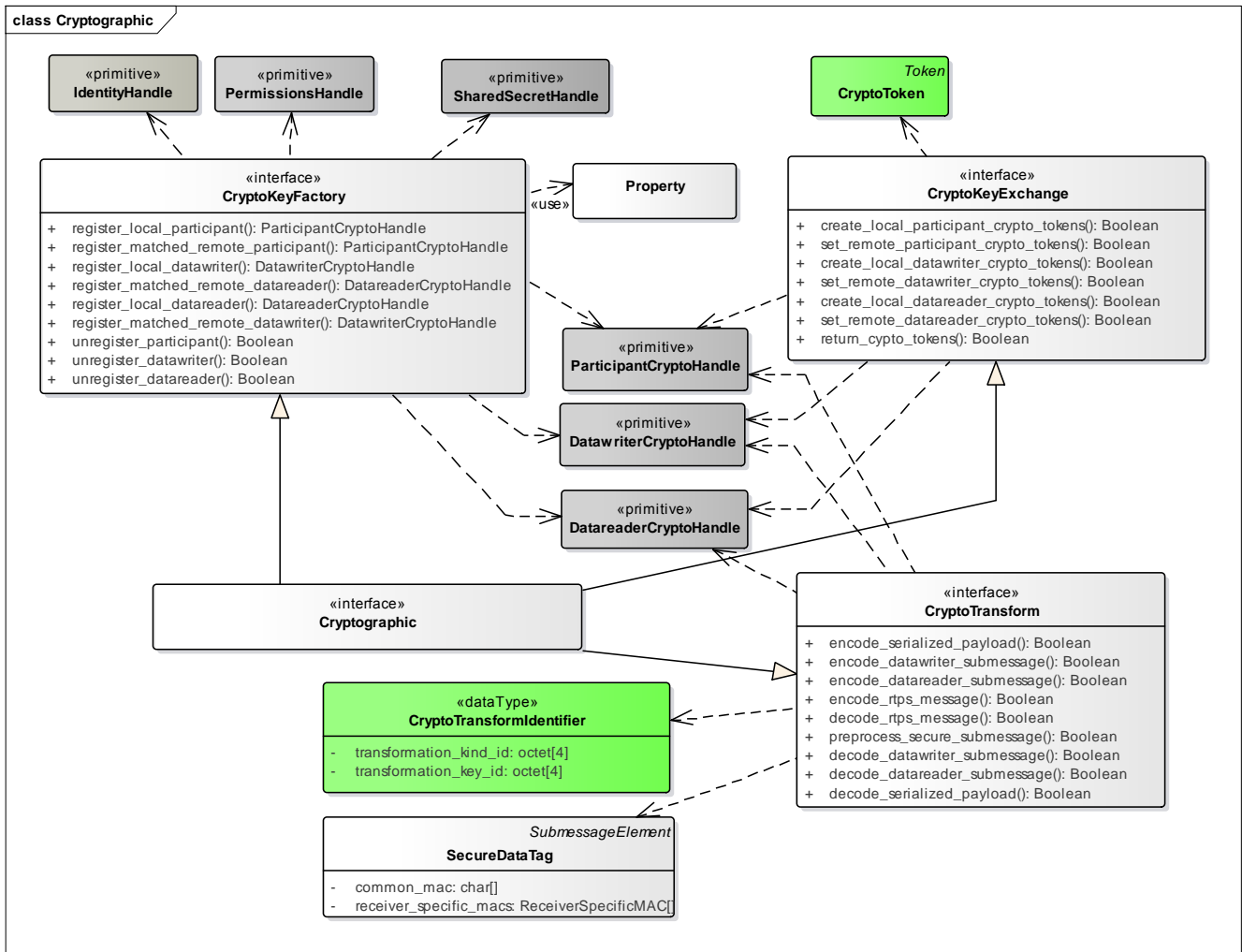
Figure 11 – Cryptographic Plugin Model

8.5.1.1   CryptoToken

This class represents a generic holder for key material. A `CryptoToken` object contains all the information necessary to construct a set of keys to be used to encrypt and/or sign plain text transforming it into cipher-text or to reverse those operations.

The format and interpretation of the `CryptoToken` depends on the implementation of the Cryptographic plugin. Each plugin implementation shall fully define itself, so that applications are able to interoperate. In general, the `CryptoToken` will contain one or more keys and any other necessary material to perform crypto-transformation and/or verification, such as, initialization vectors (IVs), salts, etc.

8.5.1.2   ParticipantCryptoHandle

The `ParticipantCryptoHandle` object is an opaque local reference that represents the key material used to encrypt and sign whole RTPS Messages. It is used by the operations `encode_rtps_message` and `decode_rtps_message`.

8.5.1.3 DatawriterCryptoHandle

The `DatawriterCryptoHandle` object is an opaque local reference that represents the key material used to encrypt and sign RTPS submessages sent from a DataWriter. This includes the RTPS submessages `Data`, `DataFrag`, `Gap`, `Heartbeat`, and `HeartbeatFrag`, as well as, the `SerializedPayload` submessage element that appears in the `Data` and `DataFrag` submessages.

It is used by the operations `encode_datawriter_submessage`, `decode_datawriter_submessage`, `encode_serialized_payload`, and `decode_serialized_payload`.

8.5.1.4 DatareaderCryptoHandle

The `DatareaderCryptoHandle` object is an opaque local reference that represents the key material used to encrypt and sign RTPS `Submessages` sent from a `DataReader`. This includes the RTPS Submessages `AckNack` and `NackFrag`.

It is used by the operations `encode_datareader_submessage`, `decode_datareader_submessage`.

8.5.1.5 CryptoTransformIdentifier

The `CryptoTransformIdentifier` object used to uniquely identify the transformation applied on the sending side (encoding) so that the receiver can locate the necessary key material to perform the inverse transformation (decoding). The generation of `CryptoTransformIdentifier` is performed by the `Cryptographic` plugin.

To enable interoperability and avoid misinterpretation of the key material, the structure of the `CryptoTransformIdentifier` is defined for all `Cryptographic` plugin implementations as follows:

```
typedef octet CryptoTransformKind[4];
typedef octet CryptoTransformKeyId[4];
struct CryptoTransformIdentifier {
    CryptoTransformKind  transformation_kind;
    CryptoTransformKeyId  transformation_key_id;
};
```

Table 34 – CryptoTransformIdentifier class

CryptoTransformIdentifier	
Attributes	
`transformation_kind`	`CryptoTransformKind`
`transformation_key_id`	`CryptoTransformKeyId`

8.5.1.5.1   Attribute: `transformation_kind`

Uniquely identifies the type of cryptographic transformation.

Values of `transformation_kind` having the first two octets set to zero are reserved by this specification, including future versions of this specification.

Implementers can use the `transformation_kind` attribute to identify non-standard cryptographic transformations. In order to avoid collisions, the first two octets in the `transformation_kind` shall be set to a registered `RTPS VendorId` [36]. The `RTPS VendorId` used shall either be one reserved to the implementer of the Cryptographic Plugin, or else the implementer of the `Cryptographic` Plugin shall secure permission from the registered owner of the `RTPS VendorId` to use it.

8.5.1.5.2 Attribute: `transformation_key_id`

Uniquely identifies the key material used to perform a cryptographic transformation within the scope of all `Cryptographic` Plugin transformations performed by the DDS `DomainParticipant` that creates the key material.

In combination with the sending `DomainParticipant` GUID, the `transformation_key_id` attribute allows the receiver to select the proper key material to decrypt/verify a message that has been encrypted and/or signed. The use of this attribute allows a receiver to be robust to dynamic changes in keys and key material in the sense that it can identify the correct key or at least detect that it does not have the necessary keys and key material.

The values of the `transformation_key_id` are defined by the `Cryptographic` plugin implementation and understood only by that plugin.

8.5.1.6 SecureSubmessageCategory_t

Enumerates the possible categories of RTPS submessages.

Table 35 – SecureSubmessageCategory_t

SecureSubmessageCategory_t	
`INFO_SUBMESSAGE`	Indicates an RTPS Info submessage: InfoSource, InfoDestination, or InfoTimestamp.
`DATAWRITER_SUMBESSAGE`	Indicates an RTPS submessage that was sent from a DataWriter: Data, DataFrag, HeartBeat, Gap.
`DATAREADER_SUMBESSAGE`	Indicates an RTPS submessage that was sent from a DataReader: AckNack, NackFrag.

8.5.1.7 CryptoKeyFactory interface

This interface groups the operations related to the creation of keys used for encryption and digital signing of both the data written by DDS applications and the RTPS submessage and message headers, used to implement the discovery protocol, distribute the DDS data, implement the reliability protocol, etc.

Table 36 – CryptoKeyFactory Interface

CryptoKeyFactory
No Attributes
Operations

register_local_partic ipant		ParticipantCryptoHandle
	participant_ide ntity	IdentityHandle
	participant_per missions	PermissionsHandle
	participant_pro perties	PropertySeq
	participant_sec urity_attribute s	ParticipantSecurityAttrib utes
	out: exception	SecurityException
register_matched_remo te_participant		ParticipantCryptoHandle
	local_participa nt_crypto_handl e	ParticipantCryptoHandle
	remote_particip ant_identity	IdentityHandle
	remote_particip ant_permissions	PermissionsHandle
	shared_secret	SharedSecretHandle
	out: exception	SecurityException
register_local_datawr iter		DatawriterCryptoHandle
	participant_cry pto	ParticipantCryptoHandle
	datawriter_prop erties	PropertySeq
	datawriter_secu rity_attributes	EndpointSecurityAttribute s
	out: exception	SecurityException

register_matched_remote_datareader		DatareaderCryptoHandle
	local_datawriter_crypto_handle	DatawriterCryptoHandle
	remote_participant_crypto	ParticipantCryptoHandle
	shared_secret	SharedSecretHandle
	relay_only	Boolean
	out: exception	SecurityException
register_local_datareader		DatareaderCryptoHandle
	participant_crypto	ParticipantCryptoHandle
	datareader_properties	PropertySeq
	datareader_security_attributes	EndpointSecurityAttributes
	out: exception	SecurityException
register_matched_remote_datawriter		DatawriterCryptoHandle
	local_datareader_crypto_handle	DatareaderCryptoHandle
	remote_participant_crypt	ParticipantCryptoHandle
	shared_secret	SharedSecretHandle
	out: exception	SecurityException
unregister_participant		Boolean
	participant_crypto_handle	ParticipantCryptoHandle
	out: exception	SecurityException
unregister_datawriter		Boolean
	datawriter_crypto_handle	DatawriterCryptoHandle
	out: exception	SecurityException

unregister_datareader		Boolean
	datareader_crypto_handle	DatareaderCryptoHandle
	out: exception	SecurityException

Registers a local `DomainParticipant` with the Cryptographic Plugin. The `DomainParticipant` must have been already authenticated and granted access to the DDS Domain. The operation shall create any necessary key material that is needed to Encrypt and Sign secure messages that are directed to other DDS `DomainParticipant` entities on the DDS Domain.

Parameter **participant_identity**: An `IdentityHandle` returned by a prior call to `validate_local_identity`. If this argument is `nil`, the operation returns `HandleNIL`.

Parameter **participant_permissions**: A `PermissionsHandle` returned by a prior call to `validate_local_permissions`. If this argument is `nil`, the operation returns `HandleNIL`.

Parameter **participant_properties**: This parameter shall contain all the properties in the `PropertyQosPolicy` of the local `DomainParticipant` whose name has the prefix "dds.sec.crypto." The purpose of this parameter is to allow configuration of the `Cryptographic` Plugin by the `DomainParticipant`, e.g., selection of the cryptographic algorithm, key size, or even setting of the key. The use of this parameter depends on the particular implementation of the plugin and shall be specified for each implementation. Properties not understood by the plugin implementation shall be silently ignored.

Parameter **participant_security_attributes**: The `ParticipantSecurityAttributes` returned by the `AccessControl` get_participant_sec_attributes operation.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `HandleNIL`.

8.5.1.7.2   Operation: register_matched_remote_participant

Registers a remote `DomainParticipant` with the `Cryptographic` Plugin. The remote `DomainParticipant` must have been already Authenticated and granted Access to the DDS Domain. The operation performs two functions:

1. It shall create any necessary key material needed to decrypt and verify the signatures of messages received from that remote `DomainParticipant` and directed to the local `DomainParticipant`.
2. It shall create any necessary key material that will be used by the local `DomainParticipant` when encrypting or signing messages that are intended only for that remote `DomainParticipant`.

Parameter **local_participant_crypto_handle**: A `ParticipantCryptoHandle` returned by a prior call to `register_local_participant`. If this argument is `nil`, the operation returns `false`.

Parameter **remote_participant_identity**: An `IdentityHandle` returned by a prior call to `validate_remote_identity`. If this argument is `nil`, the operation returns `nil`.

Parameter **participant_permissions**: A `PermissionsHandle` returned by a prior call to `validate_remote_permissions`. If this argument is `nil`, the operation returns `nil`

Parameter **shared_secret**: The `SharedSecretHandle` returned by a prior call to `get_shared_secret` as a result of the successful completion of the Authentication handshake between the local and remote `DomainParticipant` entities.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.7.3 Operation: register_local_datawriter

Registers a local `DataWriter` with the `Cryptographic` Plugin. The fact that the `DataWriter` was successfully created indicates that the `DomainParticipant` to which it belongs was authenticated, granted access to the DDS Domain, and granted permission to create the `DataWriter` on its `Topic`.

This operation shall create the cryptographic material necessary to encrypt and/or sign the data written by the `DataWriter` and returns a `DatawriterCryptoHandle` to be used for any cryptographic operations affecting messages sent or received by the `DataWriter`.

If an error occurs, this method shall return false. If it succeeds, the operation shall return an opaque handle that can be used to refer to that key material.

Parameter **participant_crypto**: A `ParticipantCryptoHandle` returned by a prior call to `register_local_participant`. It shall correspond to the `ParticipantCryptoHandle` of the `DomainParticipant` to which the `DataWriter` belongs. If this argument is `nil`, the operation returns `false`.

Parameter **local_datawriter_properties**: This parameter shall contain all the properties in the `PropertyQosPolicy` of the local `DataWriter` whose name has the prefix "`dds.sec.crypto.`" The purpose of this parameter is to allow configuration of the `Cryptographic` Plugin by the `DataWriter`, e.g., selection of the cryptographic algorithm, key size, or even setting of the key. The use of this parameter depends on the particular implementation of the plugin and shall be specified for each implementation. Properties not understood by the plugin implementation shall be silently ignored.

Parameter **datawriter_security_attributes**: The `EndpointSecurityAttributes` returned by the `AccessControl get_datawriter_sec_attributes operation`.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.7.4 Operation: register_matched_remote_datareader

Registers a remote `DataReader` with the `Cryptographic` Plugin. The remote `DataReader` shall correspond to one that has been granted permissions to match with the local `DataWriter`.

This operation shall create the cryptographic material necessary to encrypt and/or sign the RTPS submessages (`Data`, `DataFrag`, `Gap`, `Heartbeat`, `HeartbeatFrag`) sent from the local `DataWriter` to that `DataReader`. It shall also create the cryptographic material necessary to process RTPS Submessages (`AckNack`, `NackFrag`) sent from the remote `DataReader` to the `DataWriter`.

The operation shall associate the value of the *relay_only* parameter with the returned `DatawriterCryptoHandle`. This information shall be used in the generation of the `KeyToken` objects to be sent to the `DataReader`.

Parameter **local_datawriter_crypto_handle**: A `DatawriterCryptoHandle` returned by a prior call to `register_local_datawriter`. If this argument is `nil`, the operation returns `HandleNIL`.

Parameter **remote_participant_crypto**: A `ParticipantCryptoHandle` returned by a prior call to `register_matched_remote_participant`. It shall correspond to the `ParticipantCryptoHandle` of the `DomainParticipant` to which the remote `DataReader` belongs. If this argument is `nil`, the operation returns `HandleNIL`.

Parameter **shared_secret**: The `SharedSecretHandle` returned by a prior call to `get_shared_secret` as a result of the successful completion of the Authentication handshake between the local and remote `DomainParticipant` entities.

Parameter relay_only: Boolean indicating whether the cryptographic material to be generated for the remote `DataReader` shall contain everything, or only the material necessary to relay (store and forward) the information (i.e., understand the `SubmessageHeader`) without being able to decode the data itself (i.e., decode the `SecureData`).

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `HandleNIL`.

8.5.1.7.5   Operation: register_local_datareader

Registers a local `DataReader` with the `Cryptographic` Plugin. The fact that the `DataReader` was successfully created indicates that the `DomainParticipant` to which it belongs was authenticated, granted access to the DDS Domain, and granted permission to create the `DataReader` on its `Topic`.

This operation shall create the cryptographic material necessary to encrypt and/or sign the messages sent by the `DataReader` when the encryption/signature is independent of the targeted `DataWriter`.

If successful, the operation returns a `DatareaderCryptoHandle` to be used for any cryptographic operations affecting messages sent or received by the `DataWriter`.

Parameter **participant_crypto**: A `ParticipantCryptoHandle` returned by a prior call to `register_local_participant`. It shall correspond to the `ParticipantCryptoHandle` of the `DomainParticipant` to which the `DataReader` belongs. If this argument is `nil`, the operation returns `HandleNIL`.

Parameter **local_datareader_properties**: This parameter shall contain all the properties in the `PropertyQosPolicy` of the local `DataReader` whose name has the prefix "`dds.sec.crypto.`" The purpose of this parameter is to allow configuration of the `Cryptographic` Plugin by the `DataReader`, e.g., selection of the cryptographic algorithm, key size, or even setting of the key. The use of this parameter depends on the particular implementation of the plugin and shall be specified for each implementation. Properties not understood by the plugin implementation shall be silently ignored.

Parameter **datareader_security_attributes**: The `EndpointSecurityAttributes` returned by the `AccessControl get_datareader_sec_attributes operation`.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `HandleNIL`.

8.5.1.7.6 Operation: register_matched_remote_datawriter

Registers a remote `DataWriter` with the `Cryptographic` Plugin. The remote `DataWriter` shall correspond to one that has been granted permissions to match with the local `DataReader`.

This operation shall create the cryptographic material necessary to decrypt and/or verify the signatures of the RTPS submessages (`Data`, `DataFrag`, `Heartbeat`, `HeartbeatFrag`, `Gap`) sent from the remote `DataWriter` to the `DataReader`. The operation shall also create the cryptographic material necessary to encrypt and/or sign the RTPS submessages (`AckNack`, `NackFrag`) sent from the local `DataReader` to the remote `DataWriter`.

Parameter **local_datareader_crypto_handle**: A `DatareaderCryptoHandle` returned by a prior call to `register_local_datareader`. If this argument is `nil`, the operation returns `nil`.

Parameter **remote_participant_crypto**: A `ParticipantCryptoHandle` returned by a prior call to `register_matched_remote_participant`. It shall correspond to the `ParticipantCryptoHandle` of the `DomainParticipant` to which the remote `DataWriter` belongs. If this argument is `nil`, the operation returns `nil`.

Parameter **shared_secret**: The `SharedSecretHandle` returned by a prior call to `get_shared_secret` as a result of the successful completion of the Authentication handshake between the local and remote `DomainParticipant` entities.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `HandleNIL`.

8.5.1.7.7 Operation: unregister_participant

Releases the resources, associated with a `DomainParticipant` that the Cryptographic plugin maintains. After calling this function, the DDS Implementation shall not use the `participant_crypto_handle` anymore.

The DDS Implementation shall call this function when it determines that there will be no further communication with the DDS DomainParticipant associated with the `participant_crypto_handle`. Specifically, it shall be called when the application deletes a local DomainParticipant and also when the DDS Discovery mechanism detects that a matched DomainParticipant is no longer in the system.

Parameter **participant_crypto_handle**: A `ParticipantCryptoHandle` returned by a prior call to `register_local_participant`, or `register_matched_remote_participant` if this argument is `nil`, the operation returns `false`.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.7.8 Operation: unregister_datawriter

Releases the resources, associated with a `DataWriter` that the Cryptographic plugin maintains. After calling this function, the DDS Implementation shall not use the `datawriter_crypto_handle` anymore.

The DDS Implementation shall call this function when it determines that there will be no further communication with the DDS `DataWriter` associated with the `datawriter_crypto_handle`. Specifically it shall be called when the application deletes a local `DataWriter` and also when the DDS Discovery mechanism detects that a matched `DataWriter` is no longer in the system.

Parameter **datawriter_crypto_handle**: A `ParticipantCryptoHandle` returned by a prior call to `register_local_datawriter`, or `register_matched_remote_datawriter` if this argument is `nil`, the operation returns `false`.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.7.9 Operation: unregister_datareader

Releases the resources, associated with a `DataReader`, that the `Cryptographic` plugin maintains. After calling this function, the DDS Implementation shall not use the `datareader_crypto_handle` anymore.

The DDS Implementation shall call this function when it determines that there will be no further communication with the DDS `DataReader` associated with the `datareader_crypto_handle`. Specifically it shall be called when the application deletes a local `DataReader` and also when the DDS Discovery mechanism detects that a matched `DataReader` is no longer in the system.

Parameter **datareader_crypto_handle**: A `ParticipantCryptoHandle` returned by a prior call to `register_local_datareader`, or `register_matched_remote_datareader` if this argument is `nil`, the operation returns `false`.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.8 CryptoKeyExchange Interface

The key exchange interface manages the creation of keys and assist in the secure distribution of keys and key material.

Table 37 – CryptoKeyExchange Interface

CryptoKeyExchange		
No Attributes		
Operations		
`create_local_partici pant_crypto_tokens`		Boolean
	out: local_participan t_crypto_tokens	ParticipantCryptoTokenSeq
	local_participan t_crypto	ParticipantCryptoHandle
	remote_participa nt_crypto	ParticipantCryptoHandle
	out: exception	SecurityException
`set_remote_participa nt_crypto_tokens`		Boolean
	local_participan t_crypto	ParticipantCryptoHandle
	remote_participa nt_crypto	ParticipantCryptoHandle
	remote_participa nt_tokens	ParticipantCryptoTokenSeq
	out: exception	SecurityException
`create_local_datawri ter_crypto_tokens`		Boolean
	out: local_datawriter _crypto_tokens	DatawriterCryptoTokenSeq
	local_datawriter _crypto	DatawriterCryptoHandle
	remote_datareade r_crypto	DatareaderCryptoHandle
	out: exception	SecurityException

set_remote_datawrite r_crypto_tokens		Boolean
	local_datareader _crypto	DatareaderCryptoHandle
	remote_datawrite r_crypto	DatawriterCryptoHandle
	remote_datawrite r_tokens	DatawriterCryptoTokenSeq
	out: exception	SecurityException
create_local_datarea der_crypto_tokens		Boolean
	out: local_datareader _crypto_tokens	DatareaderCryptoTokenSeq
	local_datareader _crypto	DatareaderCryptoHandle
	remote_datawrite r_crypto	DatawriterCryptoHandle
	out: exception	SecurityException
set_remote_datareade r_crypto_tokens		Boolean
	local_datawriter _crypto	DatawriterCryptoHandle
	remote_datareade r_crypto	DatareaderCryptoHandle
	remote_datareade r_tokens	DatareaderCryptoTokenSeq
	out: exception	SecurityException
return_crypto_tokens		Boolean
	crypto_tokens	CryptoTokenSeq
	out: exception	SecurityException

8.5.1.8.1   Operation: create_local_participant_crypto_tokens

This operation creates a sequence of `CryptoToken` tokens containing the information needed to correctly interpret cipher text encoded using the *local_participant_crypto*. That is, the `CryptoToken` sequence contains the information needed to decrypt any data encrypted using the *local_participant_crypto*, as well as, verify any signatures produced using the *local_participant_crypto*.

The returned `CryptoToken` sequence contains opaque data, which only the plugins understand. The returned `CryptoToken` sequence is intended for transmission in "clear text" to the remote

`DomainParticipant` associated with the ***remote_participant_crypto*** so that the remote `DomainParticipant` has access to the necessary key material. For this reason, the `CryptoKeyExchange` plugin implementation may encrypt the sensitive information inside the `CryptoToken` using shared secrets and keys obtained from the ***remote_participant_crypto***. The specific ways in which this is done depend on the plugin implementation.

The DDS middleware implementation shall call this operation for each remote `DomainParticipant` that matches a local `DomainParticipant`. That is, remote participants that have been successfully authenticated and granted access by the `AccessControl` plugin. The returned `ParticipantCryptoTokenSeq` shall be sent to the remote `DomainParticipant` using the ***BuiltinParticipantVolatileMessageSecureWriter*** with kind set to `GMCLASSID_SECURITY_PARTICIPANT_CRYPTO_TOKENS` (see 7.4.3.5). The returned `ParticipantCryptoTokenSeq` sequence shall appear in the ***message_data*** attribute of the `ParticipantVolatileMessageSecure` (see 7.4.4).

Parameter local_participant_crypto_tokens (out): The returned `ParticipantCryptoTokenSeq`.

Parameter local_participant_crypto: A `ParticipantCryptoHandle`, returned by a previous call to `register_local_participant`, which corresponds to the `DomainParticipant` that will be encrypting and signing messages.

Parameter remote_participant_crypto: A `ParticipantCryptoHandle`, returned by a previous call to `register_matched_remote_participant`, that corresponds to the `DomainParticipant` that will be receiving the messages from the local `DomainParticipant` and will be decrypting them and verifying their signature.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.8.2 Operation: set_remote_participant_crypto_tokens

This operation shall be called by the DDS implementation upon reception of a message on the ***BuiltinParticipantVolatileMessageSecureReader*** with kind set to `GMCLASSID_SECURITY_PARTICIPANT_CRYPTO_TOKENS` (see 7.4.3.5).

The operation configures the `Cryptographic` plugin with the key material necessary to interpret messages encoded by the remote `DomainParticipant` associated with the ***remote_participant_crypto*** and destined to the local `DomainParticipant` associated with the ***local_participant_crypto***. The interpretation of the `CryptoToken` sequence is specific to each `Cryptographic` plugin implementation. The `CryptoToken` sequence may contain information that is encrypted and/or signed. Typical implementations of the `Cryptographic` plugin will use the previously configured shared secret associated with the local and remote `ParticipantCryptoHandle` to decode the `CryptoToken` sequence and retrieve the key material within.

Parameter remote_participant_crypto: A `ParticipantCryptoHandle`, returned by a previous call to `register_matched_remote_participant`, that corresponds to the `DomainParticipant` that will be sending the messages from the local `DomainParticipant` and will be encrypting/signing them with the key material encoded in the `CryptoToken` sequence.

Parameter local_participant_crypto: A `ParticipantCryptoHandle`, returned by a previous call to `register_local_participant`, that corresponds to the `DomainParticipant` that will be receiving messages from the remote `DomainParticipant` and will need to decrypt and/or verify their signature.

Parameter remote_participant_tokens: A `ParticipantCryptoToken` sequence received via the ***BuiltinParticipantVolatileMessageSecureReader***. The `CryptoToken` sequence shall correspond to the one returned by a call to `create_local_participant_crypto_tokens` performed by the remote `DomainParticipant` on the remote side.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.8.3   Operation: create_local_datawriter_crypto_tokens

This operation creates a `DatawriterCryptoTokenSeq` containing the information needed to correctly interpret cipher text encoded using the `local_datawriter_crypto`. That is, the `CryptoToken` sequence contains that information needed to decrypt any data encrypted using the ***local_datawriter_crypto*** as well as verify any signatures produced using the ***local_datawriter_crypto***.

The returned `CryptoToken` sequence contains opaque data, which only the plugins understand. The returned `CryptoToken` sequence shall be sent to the remote `DataReader` associated with the `remote_datareader_crypto` so that the remote `DataReader` has access to the necessary key material.

The operation shall take into consideration the value of the ***relay_only*** parameter associated with the `DatawriterCryptoHandle` (see 8.5.1.7.4) this parameter shall control whether the Tokens returned contain all the cryptographic material needed to decode/verify both the RTPS SubMessage and the CryptoContent submessage element within or just part of it.

If the value of the ***relay_only*** parameter was FALSE, the Tokens returned contain all the cryptographic material.

If the value of the ***relay_only*** parameter was TRUE, the Tokens returned contain only the cryptographic material needed to verify and decode the RTPS SubMessage but not the CryptoContent submessage element within.

The DDS middleware implementation shall call this operation for each remote `DataReader` that matches a local `DataWriter`. The returned `CryptoToken` sequence shall be sent by the DDS middleware to the remote `DataReader` using the ***BuiltinParticipantVolatileMessageSecureWriter*** with kind set to `GMCLASSID_SECURITY_DATAWRITER_CRYPTO_TOKENS` (see 7.4.3.5). The returned `DatawriterCryptoToken` shall appear in the ***message_data*** attribute of the ***ParticipantVolatileMessageSecure*** (see 7.4.4.2). The ***source_endpoint_guid*** attribute shall be set to the `GUID_t` of the local `DataWriter` and the ***destination_endpoint_guid*** attribute shall be set to the `GUID_t` of the remote `DataReader`.

Parameter local_datawriter_crypto_tokens: The returned `DatawriterCryptoTokenSeq`.

Parameter local_datawriter_crypto: A `DatawriterCryptoHandle`, returned by a previous call to `register_local_datawriter` that corresponds to the `DataWriter` that will be encrypting and signing messages.

Parameter remote_datareader_crypto: A `DatareaderCryptoHandle`, returned by a previous call to `register_matched_remote_datareader`, that corresponds to the `DataReader` that will be receiving the messages from the local `DataWriter` and will be decrypting them and verifying their signature.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.8.4   Operation: set_remote_datawriter_crypto_tokens

This operation shall be called by the DDS implementation upon reception of a message on the *BuiltinParticipantVolatileMessageSecureReader* with kind set to `GMCLASSID_SECURITY_DATAWRITER_CRYPTO_TOKENS` (see 7.4.3.5).

The operation configures the `Cryptographic` plugin with the key material necessary to interpret messages encoded by the remote `DataWriter` associated with the `remote_datawriter_crypto` and destined to the local `DataReader` associated with the `local_datareader_crypto`. The interpretation of the `DatawriterCryptoTokenSeq` sequence is specific to each `Cryptographic` plugin implementation. The `CryptoToken` sequence may contain information that is encrypted and/or signed. Typical implementations of the `Cryptographic` plugin will use the previously configured shared secret associated with the remote `DatawriterCryptoHandle` and local `DatareaderCryptoHandle` to decode the `CryptoToken` sequence and retrieve the key material within.

Parameter remote_datawriter_crypto: A `DatawriterCryptoHandle`, returned by a previous call to `register_matched_remote_datawriter`, that corresponds to the `DataWriter` that will be sending the messages to the local `DataReader` and will be encrypting/signing them with the key material encoded in the `CryptoToken`.

Parameter local_datareader_crypto: A `DatareaderCryptoHandle`, returned by a previous call to `register_local_datareader`, that corresponds to the `DataReader` that will be receiving messages from the remote `DataWriter` and will need to decrypt and/or verify their signature.

Parameter remote_datawriter_tokens: A `CryptoToken` sequence received via the *BuiltinParticipantVolatileMessageSecureReader*. The `DatawriterCryptoToken` shall correspond to the one returned by a call to `create_local_datawriter_crypto_tokens` performed by the remote `DataWriter` on the remote side.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.8.5   Operation: create_local_datareader_crypto_tokens

This operation creates a `DatareaderCryptoTokenSeq` containing the information needed to correctly interpret cipher text encoded using the *local_datareader_crypto*. That is, the `CryptoToken` sequence contains that information needed to decrypt any data encrypted using the *local_datareader_crypto* as well as verify any signatures produced using the *local_datareader_crypto*.

The returned `CryptoToken` sequence contains opaque data, which only the plugins understand. The returned `CryptoToken` sequence shall be sent to the remote `DataWriter` associated with the *remote_datawriter_crypto* so that the remote `DataWriter` has access to the necessary key material.

For this reason, the `CryptoKeyExchange` plugin implementation may encrypt the sensitive information inside the `CryptoToken` sequence using shared secrets and keys obtained from the **remote_datawriter_crypto**. The specific ways in which this is done depend on the plugin implementation.

The DDS middleware implementation shall call this operation for each remote `DataWriter` that matches a local `DataReader`. The returned `DatareaderCryptoTokenSeq` shall be sent by the DDS middleware to the remote `DataWriter` using the **BuiltinParticipantVolatileMessageSecureWriter** with kind set to `GMCLASSID_SECURITY_DATAREADER_CRYPTO_TOKENS` (see 7.4.4.2). The returned `DatareaderCryptoTokenSeq` shall appear in the **message_data** attribute of the `ParticipantVolatileMessageSecure` (see 7.4.4.2). The **source_endpoint_guid** attribute shall be set to the `GUID_t` of the local `DataReader` and the **destination_endpoint_guid** attribute shall be set to the `GUID_t` of the remote `DataWriter`.

Parameter local_datareader_crypto_tokens (out): The returned `DatareaderCryptoTokenSeq`.

Parameter local_datareader_crypto: A `DatareaderCryptoHandle`, returned by a previous call to `register_local_datareader`, that corresponds to the `DataReader` that will be encrypting and signing messages.

Parameter remote_datawriter_crypto: A `DatawriterCryptoHandle`, returned by a previous call to `register_matched_remote_datawriter`, that corresponds to the `DataWriter` that will be receiving the messages from the local `DataReader` and will be decrypting them and verifying their signature.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.8.6 Operation: set_remote_datareader_crypto_tokens

This operation shall be called by the DDS implementation upon reception of a message on the **BuiltinParticipantVolatileMessageSecureReader** with kind set to `GMCLASSID_SECURITY_DATAREADER_CRYPTO_TOKENS` (see 7.4.4.2).

The operation configures the `Cryptographic` plugin with the key material necessary to interpret messages encoded by the remote `DataReader` associated with the **remote_datareader_crypto** and destined to the local `DataWriter` associated with the **local_datawriter_crypto**. The interpretation of the `DatareaderCryptoTokenSeq` is specific to each `Cryptographic` plugin implementation. The `CryptoToken` sequence may contain information that is encrypted and/or signed. Typical implementations of the `Cryptographic` plugin will use the previously configured shared secret associated with the remote `DatareaderCryptoHandle` and local `DatawriterCryptoHandle` to decode the `CryptoToken` sequence and retrieve the key material within.

Parameter remote_datareader_crypto: A `DatareaderCryptoHandle`, returned by a previous call to `register_matched_remote_datareader`, that corresponds to the `DataReader` that will be sending the messages to the local `DataWriter` and will be encrypting/signing them with the key material encoded in the `CryptoToken` sequence.

Parameter local_datawriter_crypto: A `DatawriterCryptoHandle` returned by a previous call to `register_local_datawriter`, that corresponds to the `DataWriter` that will be receiving messages from the remote `DataReader` and will need to decrypt and/or verify their signature.

Parameter remote_datareader_tokens: A `CryptoToken` sequence received via the ***BuiltinParticipantVolatileMessageSecureReader***. The `DatareaderCryptoToken` shall correspond to the one returned by a call to `create_local_datareader_crypto_tokens` performed by the remote `DataReader` on the remote side.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.8.7 Operation: return_crypto_tokens

Returns the tokens in the `CryptoToken` sequence to the plugin so the plugin can release any information associated with it.

Parameter crypto_tokens: Contains `CryptoToken` objects issued by the plugin on a prior call to one of the following operations:

- create_local_participant_crypto_tokens
- create_local_datawriter_crypto_tokens
- create_local_datareader_crypto_tokens

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9 CryptoTransform interface

This interface groups the operations related to encrypting/decrypting, as well as, computing and verifying both message digests (hashes) and Message Authentication Codes (MAC).

MACs may be used to verify both the (data) integrity and the authenticity of a message. The computation of a MAC (also known as a keyed cryptographic hash function), takes as input a secret key and an arbitrary-length message to be authenticated, and outputs a MAC. The MAC value protects both a message's data integrity, as well as, its authenticity by allowing verifiers (who also possess the secret key) to detect any changes to the message content.

A Hash-based Message Authentication Code (HMAC) is a specialized way to compute MACs. While an implementation of the plugin is not forced to use HMAC, and could use other MAC algorithms, the API is chosen such that plugins can implement HMAC if they so choose.

The operations in the `CryptoTransform` Plugin are defined to be quite generic, taking an input byte array to transform and producing the transformed array of bytes as an output. The DDS implementation is only responsible for calling the operations in the CryptoTransform plugin at the appropriate times as it generates and processes the `RTPS` messages, substitutes the input bytes with the transformed bytes produced by the `CryptoTransform` operations, and proceeds to generate/send or process the RTPS message as normal but with the replaced bytes. The decision of the kind of transformation to perform (encrypt and/or produce a digest and/or a MAC and/or signature) is left to the plugin implementation.

Table 38 – CryptoTransform interface

CryptoTransform		
No Attributes		
Operations		
`encode_serialized_payload`		Boolean
	`out: encoded_buffer`	`octet[]`
	`out: extra_inline_qos`	`octet[]`
	`plain_buffer`	`octet[]`
	`sending_datawriter_crypto`	`DatawriterCryptoHandle`
	`out: exception`	`SecurityException`
`encode_datawriter_submessage`		Boolean
	`out: encoded_rtps_submessage`	`octet[]`
	`plain_rtps_submessage`	`octet[]`
	`sending_datawriter_crypto`	`DatawriterCryptoHandle`
	`receiving_datareader_crypto_list`	`DatareaderCryptoHandle[]`
	`inout: receiving_datareader_crypto_list_index`	`long`
	`out: exception`	`SecurityException`

encode_datareader_su bmessage		Boolean
	out: encoded_rtps_sub message	octet[]
	plain_rtps_subme ssage	octet[]
	sending_dataread er_crypto	DatareaderCryptoHandle
	receiving_datawr iter_crypto_list	DatawriterCryptoHandle[]
	out: exception	SecurityException
encode_rtps_message		Boolean
	out: encoded_rtps_mes sage	octet[]
	plain_rtps_messa ge	octet[]
	sending_particip ant_crypto	ParticipantCryptoHandle
	receiving_partic ipant_crypto_lis t	ParticipantCryptoHandle[]
	inout: receiving_partic ipant_crypto_lis t_index	long
	out: exception	SecurityException
decode_rtps_message		Boolean
	out: plain_buffer	octet[]
	encoded_buffer	octet[]
	receiving_partic ipant_crypto	ParticipantCryptoHandle
	sending_particip ant_crypto	ParticipantCryptoHandle
	out: exception	SecurityException

preprocess_secure_su bmsg		Boolean
	out: datawriter_crypt o	DatawriterCryptoHandle
	out: datareader_crypt o	DatareaderCryptoHandle
	out: secure_submessag e_category	DDS_SecureSumessageCatego ry_t
	in: encoded_rtps_sub message	octet[]
	receiving_partic ipant_crypto	ParticipantCryptoHandle
	sending_particip ant_crypto	ParticipantCryptoHandle
	out: exception	SecurityException
decode_datawriter_su bmessage		Boolean
	out: plain_rtps_subme ssage	octet[]
	encoded_rtps_sub message	octet[]
	receiving_datare ader_crypto	DatareaderCryptoHandle
	sending_datawrit er_crypto	DatawriterCryptoHandle
	out: exception	SecurityException

decode_datareader_su bmessage		Boolean
	out: plain_rtps_subme ssage	octet[]
	encoded_rtps_sub message	octet[]
	receiving_datawr iter_crypto	DatawriterCryptoHandle
	sending_dataread er_crypto	DatareaderCryptoHandle
	out: exception	SecurityException
decode_serialized_pa yload		Boolean
	out: plain_buffer	octet[]
	encoded_buffer	octet[]
	inline_qos	octet[]
	receiving_datare ader_crypto	DatareaderCryptoHandle
	sending_datawrit er_crypto	DatawriterCryptoHandle
	out: exception	SecurityException

8.5.1.9.1 Operation: encode_serialized_payload

This operation shall be called by the DDS implementation as a result of the application calling the write operation on the DataWriter associated with the `DatawriterCryptoHandle` specified in the ***sending_datawriter_crypto*** parameter.

The operation receives the data written by the `DataWriter` in serialized form wrapped inside the RTPS `SerializedPayload` submessage element and shall output an RTPS `CryptoContent` submessage element and a ***extra_inline_qos*** containing InlineQos formatted as a `ParameterList`, see section 7.3.1.

If the returned ***extra_inline_qos*** is not empty, the parameters contained shall be added to the list of ***inlineQos*** parameters present in the (`Data` or `DataFrag`) submessage. If the (`Data` or `DataFrag`) submessage did not already have an ***inlineQos***, then the ***inlineQos*** submessage element shall be added and the submessage flags modified accordingly.

The DDS implementation shall call this operation for all outgoing RTPS Submessages with submessage kind `Data` and `DataFrag`. The DDS implementation shall substitute the

`SerializedPayload` submessage element within the aforementioned RTPS submessages with the `CryptoContent` produced by this operation.

The implementation of `encode_serialized_payload` can perform any desired cryptographic transformation of the `SerializedPayload` using the key material in the `sending_datawriter_crypto,` including encryption, addition of a MAC, and/or signature. The `encode_serialized_payload` shall include in the *extra_inline_qos* or the `CryptoContent` the `CryptoTransformIdentifier` and the additional information needed to identify the key used and decode the `CryptoContent` submessage element.
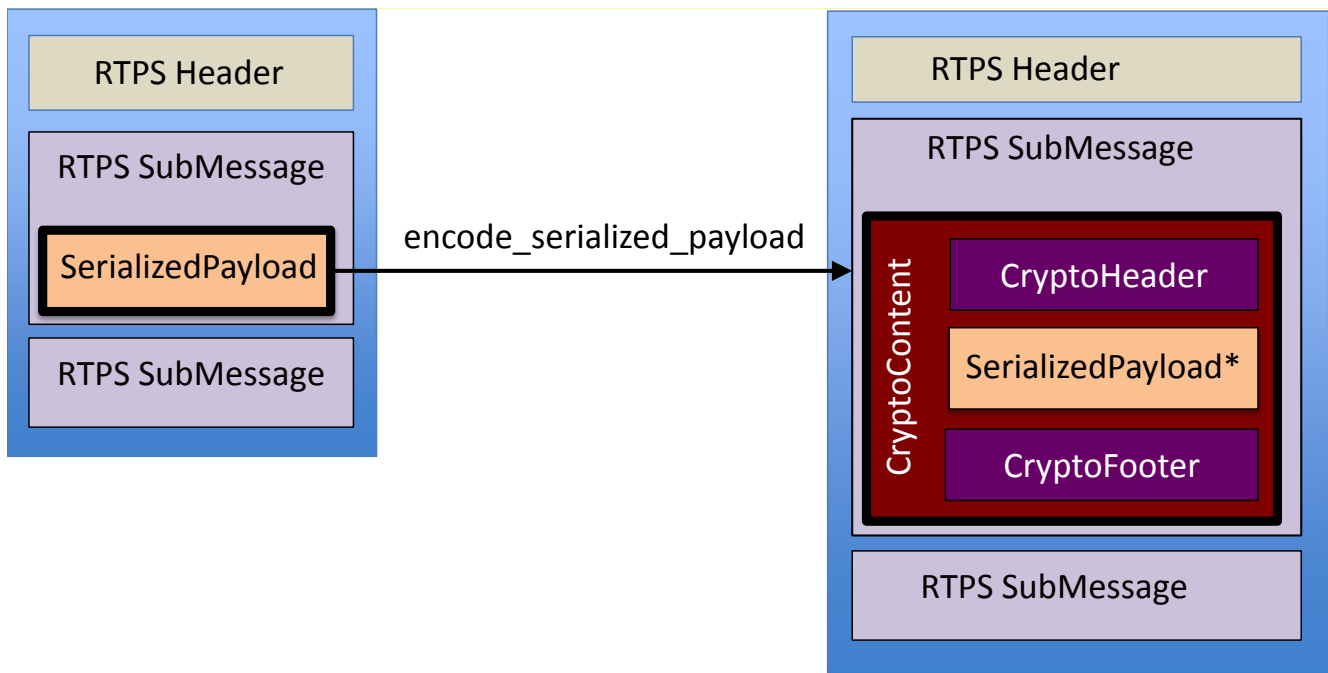
Figure 12 – Effect of encode_serialized_payload within an RTPS message

If an error occurs, this method shall return `false`.

Parameter *encoded_buffer*: The output containing the CryptoContent RTPS submessage element, which shall be used to replace the input *plain_buffer*.

Parameter *extra_inline_qos*: The output containing additional parameters to be added to the inlineQos `ParamaterList` in the submessage.

Parameter **plain_buffer**: The input containing the `SerializedPayload` RTPS submessage element.

Parameter **sending_datawriter_crypto**: The `DatawriterCryptoHandle` returned by a previous call to `register_local_datawriter` for the `DataWriter` that wrote the `SerializedPayload`.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9.2   Operation: encode_datawriter_submessage

This operation shall be called by the DDS implementation whenever it has constructed an RTPS submessage of kind `Data`, `DataFrag`, `Gap`, `Heartbeat`, or `HeartbeatFrag`.

The operation receives the `DatawriterCryptoHandle` of the `DataWriter` that is sending the submessage, as well as, a list of `DatareaderCryptoHandle` corresponding to all the `DataReader` entities to which the submessage is being sent.

In the case of ***BuiltinParticipantVolatileMessageSecureWriter*** (identified through the `DatawriterCryptoHandle`), the `DatareaderCryptoHandle` list has ONE element containing `KxKey` material derived from the `SharedSecret` as described in 9.5.2.1.2.

The operation receives the complete RTPS submessage as it would normally go onto the wire in the parameter ***rtps_submessage*** and shall output one or more RTPS `Submessages` in the output parameter ***encoded_rtps_submessage***. The DDS implementation shall substitute the original RTPS submessage that was passed in the ***rtps_submessage*** with the RTPS `Submessages` returned in the ***encoded_rtps_submessage*** output parameter in the construction of the RTPS message that is eventually sent to the intended recipients.

The implementation of `encode_datawriter_submessage` can perform any desired cryptographic transformation of the RTPS `Submessage` using the key material in the ***sending_datawriter_crypto***; it can also add one or more MACs and/or signatures. The fact that the cryptographic material associated with the list of intended `DataReader` entities is passed in the parameter ***receiving_datareader_crypto_list*** allows the plugin implementation to include MACs that may be computed differently for each `DataReader`.

The implementation of `encode_datawriter_submessage` shall include, within the RTPS `Submessages`, the `CryptoTransformIdentifier` containing any additional information necessary for the receiving plugin to identify the `DatawriterCryptoHandle` associated with the DataWriter that sent the message, as well as, the `DatareaderCryptoHandle` associated with the DataReader that is meant to process the submessage. How this is done depends on the plugin implementation.

A typical implementation of `encode_datawriter_submessage` may output a `SecurePrefixSubMsg` followed by a `SecureBodySubMsg`, followed by a `SecurePostfixSubMsg`.

If an error occurs, this method shall return `false`.
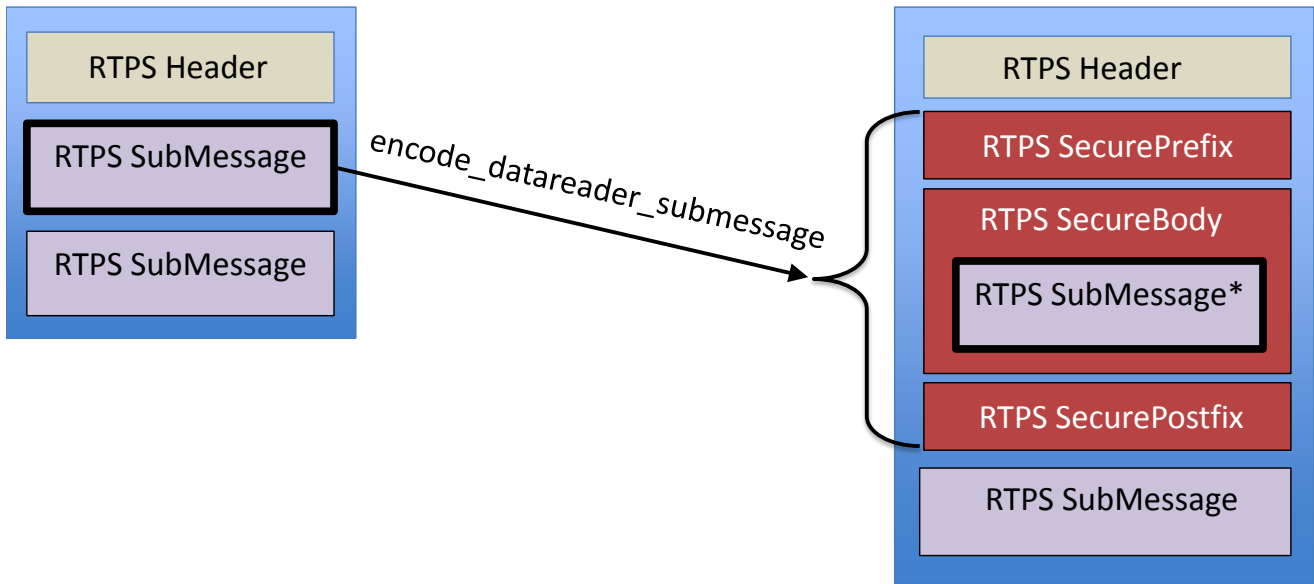
Figure 13 – Effect of encode_datawriter_submessage within an RTPS message

Parameter **encoded_rtps_submessage**: The output containing one or more RTPS submessages, which shall be used to replace the input *rtps_submessage*.

Parameter **plain_rtps_submessage**: The input containing the RTPS submessage created by a DataWriter. This submessage will be one of following kinds: `Data`, `DataFrag`, `Gap`, `Heartbeat`, and `HeartbeatFrag`.

Parameter **sending_datawriter_crypto**: The `DatawriterCryptoHandle` returned by a previous call to register_local_datawriter for the DataWriter whose GUID is inside the `rtps_submessage`.

Parameter **receiving_datareader_crypto_list**: The list of `DatareaderCryptoHandle` returned by previous calls to `register_matched_remote_datareader` for the DataReader entities to which the submessage will be sent.

Parameter **receiving_datareader_crypto_list_index**: Index to the first element of the *receiving_datareader_crypto_list* that should be used. This parameter allows the `encode_datawriter_submessage` operation to be invoked multiple times for a given *plain_rtps_submessage*, iterating over elements in the receiving_*datawriter_crypto_list*. Each iteration prepares the *encoded_rtps_submessage* for a different set of data readers and advances the *receiving_datareader_crypto_list_index*.

The *receiving_datareader_crypto_list_index* shall be set to 0 to start the iteration on a *plain_rtps_submessage*. Subsequent calls may use a non-zero value of the index. If the index is non-zero, then the *plain_rtps_submessage* shall be set to the empty sequence and the *encoded_rtps_submessage* shall be the one returned by a previous call to the `encode_datawriter_submessage`. The calls with non-zero values of the *receiving_datareader_crypto_list_index* modify the *encoded_rtps_submessage*, replacing the receiver-specifc parts of the *encoded_rtps_submessage*.

The operation fills the *receiving_datawriter_crypto_list_index* with the next index to use in subsequent calls to `encode_datawriter_submessage`. The value *receiving_datawriter_crypto_list_index* = Length(*receiving_datawriter_crypto_list*) indicates that the iteration over the *receiving_datawriter_crypto_list* is complete.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9.3   Operation: encode_datareader_submessage

This operation shall be called by the DDS implementation whenever it has constructed an RTPS submessage of kind `AckNack` or `NackFrag`.

The operation receives the `DatareaderCryptoHandle` of the DataReader that is sending the submessage, as well as, a list of `DatawriterCryptoHandle` corresponding to all the DataWriter entities to which the submessage is being sent.

In the case of ***BuiltinParticipantVolatileMessageSecureReader*** (identified through the `DatawriterCryptoHandle`), the `DatawriterCryptoHandle` list has ONE element containing `KxKey` material derived from the `SharedSecret` as described in 9.5.2.1.2.

The operation receives the complete RTPS submessage as it would normally go onto the wire in the parameter ***rtps_submessage*** and shall output one or more RTPS `Submessages` in the output parameter ***encoded_rtps_submessage***. The DDS implementation shall substitute the original RTPS submessage that was passed in the ***rtps_submessage*** with the `Submessages` returned in the ***encoded_rtps_submessage*** output parameter in the construction of the RTPS message that is eventually sent to the intended recipients.

The implementation of `encode_datareader_submessage` can perform any desired cryptographic transformation of the RTPS Submessage using the key material in the `sending_datareader_crypto`, it can also add one or more MACs, and/or signatures. The fact that the cryptographic material associated with the list of intended `DataWriter` entities is passed in the parameter `receiving_datawriter_crypto_list` allows the plugin implementation to include one of MAC that may be computed differently for each `DataWriter`.

The implementation of `encode_datareader_submessage` shall include within the ***encoded_rtps_submessage*** the `CryptoTransformIdentifier` containing any additional information necessary for the receiving plugin to identify the `DatareaderCryptoHandle` associated with the DataReader that sent the message as well as the `DatawriterCryptoHandle` associated with the DataWriter that is meant to process the submessage. How this is done depends on the plugin implementation.

A typical implementation of `encode_datareader_submessage` may output a `SecurePrefixSubMsg` followed by a `SecureBodySubMsg`, followed by a `SecurePostfixSubMsg`.

If an error occurs, this method shall return `false`.
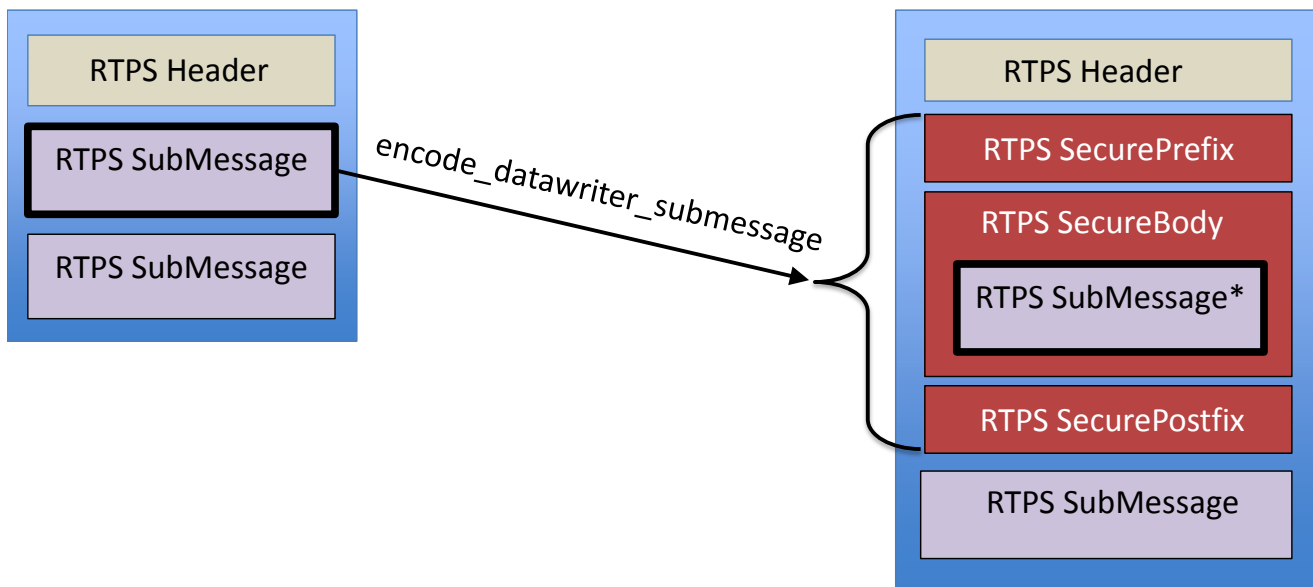
Figure 14 – Effect of encode_datareader_submessage within an RTPS message

Parameter **encoded_rtps_submessage**: The output containing one or more RTPS submessages, which shall be used to replace the input `rtps_submessage`.

Parameter **plain_rtps_submessage**: The input containing the RTPS submessage created by a DataReader. This submessage will be one of following kinds: `AckNack, NackFrag.`

Parameter **sending_datareader_crypto**: The `DatareaderCryptoHandle` returned by a previous call to register_local_datareader for the DataReader whose GUID is inside the `rtps_submessage`.

Parameter **receiving_datawriter_crypto_list**: The list of `DatawriterCryptoHandle` returned by previous calls to `register_matched_remote_datawriter` for the DataWriter entities to which the submessage will be sent.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9.4   Operation: encode_rtps_message

This operation shall be called by the DDS implementation whenever it has constructed an RTPS message prior to sending it on the wire.

The operation receives the `ParticipantCryptoHandle` of the DomainParticipant that is sending the submessage, as well as, a list of `ParticipantCryptoHandle` corresponding to all the DomainParticipant entities to which the submessage is being sent.

The operation receives the complete RTPS message as it would normally go onto the wire in the parameter *plain_rtps_message* and shall also output an RTPS message in the output parameter *encoded_rtps_message*. The DDS implementation shall substitute the original RTPS message that was passed in the *plain_rtps_message* with the *encoded_rtps_message* returned by this operation and proceed to send it to the intended recipients.

This operation may optionally not perform any transformation of the input RTPS message. In this case, the operation shall return false but not set the `exception` object. In this situation the DDS implementation shall send the original RTPS message.

The implementation of `encode_rtps_message` may perform any desired cryptographic transformation of the whole RTPS Message using the key material in the `sending_participant_crypto`, it can also add one or more MACs, and/or signatures. The fact that the cryptographic material associated with the list of intended DataWriter entities is passed in the parameter `receiving_participant_crypto_list` allows the plugin implementation to include one of MAC that may be computed differently for each destination `DomainParticipant`.

The implementation of `encode_rtps_message` shall include within the *encoded_rtps_message* the `CryptoTransformIdentifier` containing any additional information beyond the one shared via the `CryptoToken` that would be needed to identify the key used and decode the *encoded_rtps_message* back into the original RTPS message.

A typical implementation of `encode_rtps_message` to provide authentication only may output the `RTPS Header` followed by a `SecureRTPSPrefixSubMsg` followed by a `InfoSourceSubMsg` (containing the information in the original RTPS Header so it can be authenticated), followed by the submessages included in the input *plain_rtps_message*, followed by a `SecureRTPSPostfixSubMsg`.

If an error occurs, this method shall return `false` and set the `exception` object.
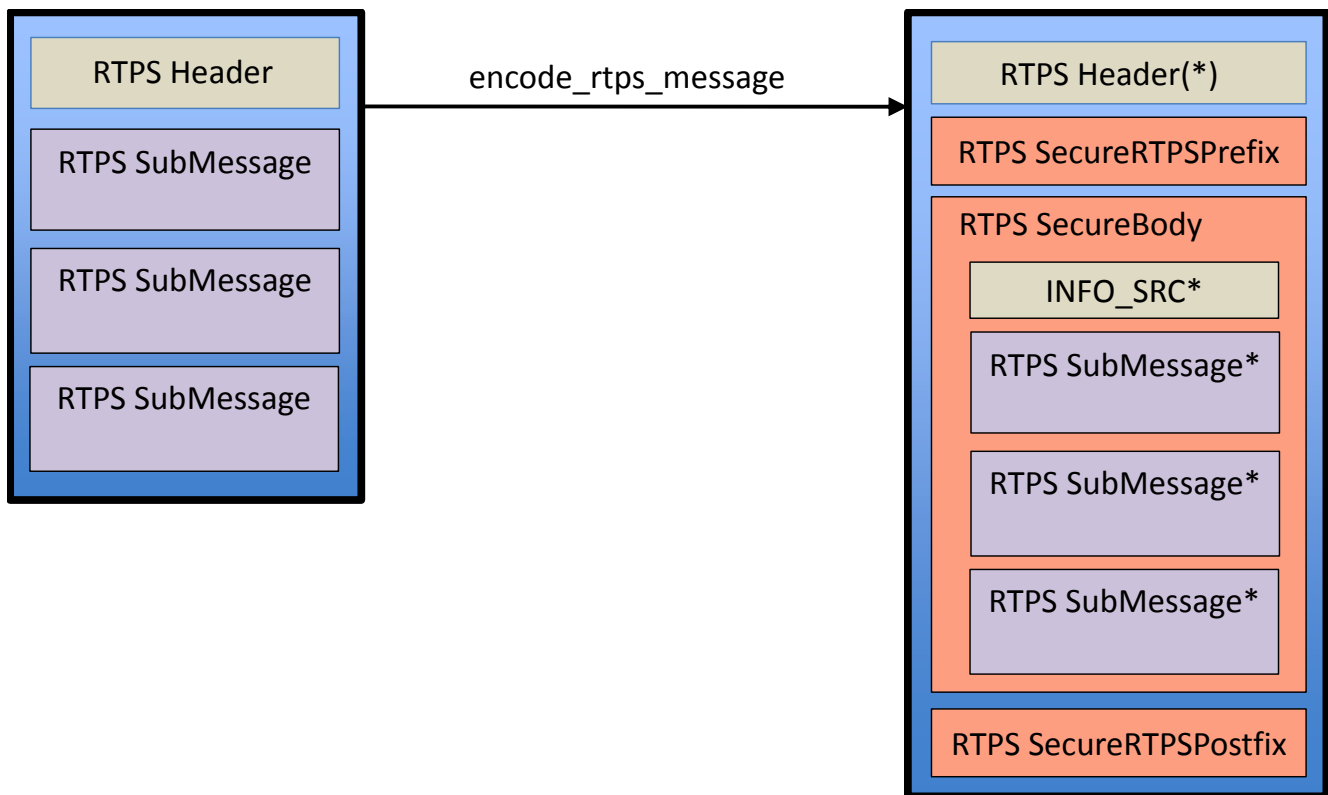
Figure 15 – Possible effect of encode_rtps within an RTPS message

Parameter **encoded_rtps_message**: The output containing the encoded RTPS message.

Parameter **plain_rtps_message**: The input containing the RTPS messages the DDS implementation intended to send.

Parameter **sending_participant_crypto**: The `ParticipantCryptoHandle` returned by a previous call to register_local_participant for the DomainParticipant whose GUID is inside the RTPS Header.

Parameter **receiving_participant_crypto_list**: The list of `ParticipantCryptoHandle` returned by previous calls to `register_matched_remote_participant` for the DomainParticipant entities to which the message will be sent.

Parameter **receiving_participant_crypto_list_index**: Index to the first element of the *receiving_participant_crypto_list* that should be used. This parameter allows the `encode_rtps_message` operation to be invoked multiple times for a given *plain_rtps_message*, iterating over elements in the receiving *receiving_participant_crypto_list*. Each iteration prepares the *encoded_rtps_message* for a different set of receiving domain participants and advances the *receiving_participant_crypto_list_index*.

The *receiving_participant_crypto_list_index* shall be set to 0 to start the iteration on a *plain_rtps_message*. Subsequent calls may use a non-zero value of the index. If the index is non-zero, then the *plain_rtps_message* shall be set to the empty sequence and the *encoded_rtps_message* shall be the one returned by a previous call to the `encode_rtps_message`. The calls with non-zero values of the *receiving_participant_crypto_list_index* modify the *encoded_rtps_message*, replacing the receiver-specifc parts of the *encoded_rtps_message*.

The operation fills the *receiving_participant_crypto_list_index* with the next index to use in subsequent calls to `encode_rtps_message`. The value *receiving_participant_crypto_list_index* = Length(*receiving_participant_crypto_list*) indicates that the iteration over the *receiving_participant_crypto_list* is complete.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9.5   Operation: decode_rtps_message

This operation shall be called by the DDS implementation whenever it receives an RTPS message prior to parsing it.

This operation shall reverse the transformation performed by the `encode_rtps_message` operation, decrypting the content if appropriate and verifying any MACs or digital signatures that were produced by the `encode_rtps_message` operation.

If an error occurs, this method shall return an exception.
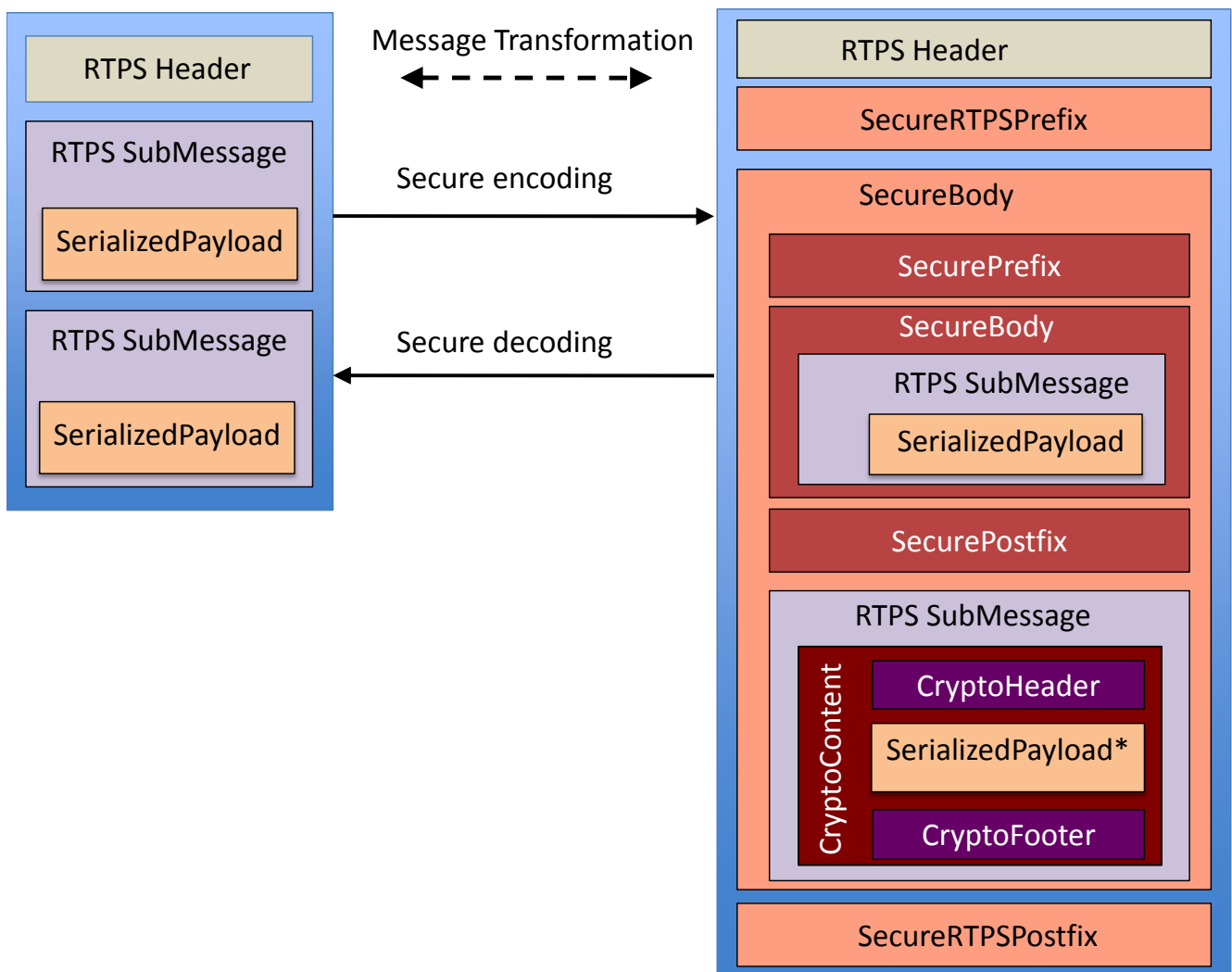
Figure 16 – Possible effect of decode_rtps within an RTPS message

Parameter **plain_rtps_message**: The output containing the decoded RTPS message. The output message shall contain the original RTPS message.

Parameter **encoded_rtps_message**: The input containing the encoded RTPS message the DDS implementation received.

Parameter **receiving_participant_crypto**: The `ParticipantCryptoHandle` returned by previous calls to `register_local_participant` for the `DomainParticipant` entity that received the RTPS message.

Parameter **sending_participant_crypto**: The `ParticipantCryptoHandle` returned by a previous call to `register_matched_remote_participant` for the `DomainParticipant` that sent the RTPS message whose GUID is inside the RTPS Header.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9.6 Operation: preprocess_secure_submsg

This operation shall be called by the DDS implementation as a result of a `DomainParticipant` receiving an RTPS.

The purpose of the operation is to determine whether the secure submessage was produced as a result of a call to `encode_datawriter_submessage` or a call to `encode_datareader_submessage`, and to retrieve the appropriate `DatawriterCryptoHandle` and `DatareaderCryptoHandle` needed to decode the submessage.

If the operation returns successfully, the DDS implementation shall call the appropriate decode operation based on the returned `SecureSubmessageCategory_t`:

- If the returned `SecureSubmessageCategory_t` equals DATAWRITER_SUBMESSAGE, then the DDS Implementation shall call `decode_datawriter_submessage`.
- If the returned `SecureSubmessageCategory_t` equals DATAREADER_SUBMESSAGE, then the DDS Implementation shall call `decode_datareader_submessage`.
- If the returned `SecureSubmessageCategory_t` equals INFO_SUBMESSAGE, then the DDS Implementation proceeds normally to process the submessage without further decoding.

Parameter **secure_submessage_category**: Output `SecureSubmessageCategory_t`. It shall be set to `DATAWRITER_SUBMESSAGE` if the `SecurePrefixSubMsg` was created by a call to `encode_datawriter_submessage` or set to `DATAREADER_SUBMESSAGE` if the `SecurePrefixSubMsg` was created by a call to `encode_datareader_submessage`. If none of these conditions apply, the operation shall return false.

Parameter **datawriter_crypto**: Output `DatawriterCryptoHandle`. The setting depends on the returned value of `secure_submessage_category`:

- If `secure_submessage_category` is `DATAWRITER_SUBMESSAGE`, the `datawriter_crypto` shall be the `DatawriterCryptoHandle` returned by a previous call to `register_matched_remote_datawriter` for the DataWriter that wrote the RTPS Submessage.
- If `secure_submessage_category` is `DATAREADER_SUBMESSAGE`, the `datawriter_crypto` shall be the `DatawriterCryptoHandle` returned by a previous call to `register_local_datawriter` for the DataWriter that is also the destination of the RTPS Submessage.

Parameter **datareader_crypto**: Output `DatareaderCryptoHandle`. The setting depends on the returned value of `secure_submessage_category`:

- If `secure_submessage_category` is `DATAWRITER_SUBMESSAGE`, the `datareader_crypto` shall be the `DatareaderCryptoHandle` returned by a previous call to `register_local_datareader` for the DataReader that is the destination of the RTPS Submessage.
- If `secure_submessage_category` is `DATAREADER_SUBMESSAGE`, the `datareader_crypto` shall be the `DatareaderCryptoHandle` returned by a previous call to `register_matched_remote_datareader` for the DataReader that wrote the RTPS Submessage.

Parameter **encoded_rtps_message**: The input containing the received RTPS message.

Parameter **receiving_participant_crypto**: The `ParticipantCryptoHandle` returned by previous calls to `register_local_participant` for the DomainParticipant that received the RTPS message.

Parameter **sending_participant_crypto**: The `ParticipantCryptoHandle` returned by a previous call to `register_matched_remote_participant` for the DomainParticipant whose GUID is inside the RTPS Header.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9.7 Operation: decode_datawriter_submessage

This operation shall be called by the DDS implementation as a result of receiving a `SecurePrefixSubMsg` whenever the preceding call to `preprocess_secure_submessage` identified the `SecureSubmessageCategory_t` as DATAWRITER_SUBMESSAGE.

This operation shall reverse the transformation performed by the `encode_datawriter_submessage` operation, decrypting the content if appropriate and verifying any MACs or digital signatures that were produced by the `encode_datawriter_submessage` operation.

The DDS implementation shall substitute the RTPS `SecurePrefixSubMsg` and any associated submessages following (for example, `SecureBodySubMsg` and `SecurePostfixSubMsg`) within the received submessages with the RTPS `Submessage` produced by this operation.

If an error occurs, this method shall return `false`.
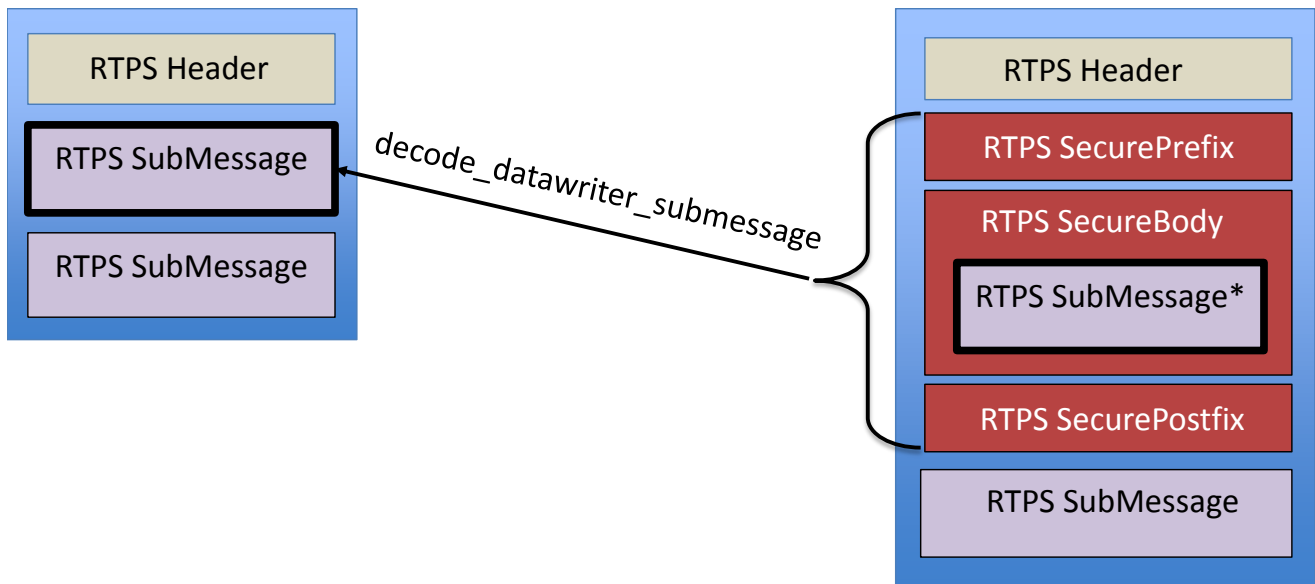
Figure 17 – Effect of decode_datawriter_submessage within an RTPS message

Parameter **plain_rtps_submessage**: The output containing the RTPS submessage created by a DataWriter. This submessage will be one of following kinds: `Data`, `DataFrag`, `Gap`, `Heartbeat`, and `HeartbeatFrag`.

Parameter **encoded_rtps_submessage**: The input containing the RTPS `SecurePrefixSubMsg` and any associated submessages following (for example, `SecureBodySubMsg` and

SecurePostfixSubMsg), which were created by a call to
`encode_datawriter_submessage`.

Parameter **receiving_datareader_crypto**: The `DatareaderCryptoHandle` returned by the preceding call to `preprocess_secure_submessage` performed on the received `SecurePrefixSubMsg`. It shall contain the `DatareaderCryptoHandle` corresponding to the DataReader that is receiving the RTPS Submessage.

Parameter **sending_datawriter_crypto**: The `DatawriterCryptoHandle` returned by the preceding call to `preprocess_secure_submsg` performed on the received `SecurePrefixSubMsg`. It shall contain the `DatawriterCryptoHandle` corresponding to the DataWriter that is sending the RTPS Submessage.

Parameter **exception**: A `SecurityException` object, which provides details in case this operation returns `false`.

8.5.1.9.8   Operation: decode_datareader_submessage

This operation shall be called by the DDS implementation as a result of receiving a `SecurePrefixSubMsg` whenever the preceding call `to preprocess_secure_submessage` identified the `SecureSubmessageCategory_t` as DATAREADER_SUBMESSAGE.

This operation shall reverse the transformation performed by the `encode_datareader_submessage` operation, decrypting the content if appropriate and verifying any MACs or digital signatures that were produced by the `encode_datareader_submessage` operation.

The DDS implementation shall substitute the RTPS `SecurePrefixSubMsg` and any associated submessages following (for example, `SecureBodySubMsg` and `SecurePostfixSubMsg`) within the received submessages with the RTPS `Submessage` produced by this operation.

If an error occurs, this method shall return `false`.
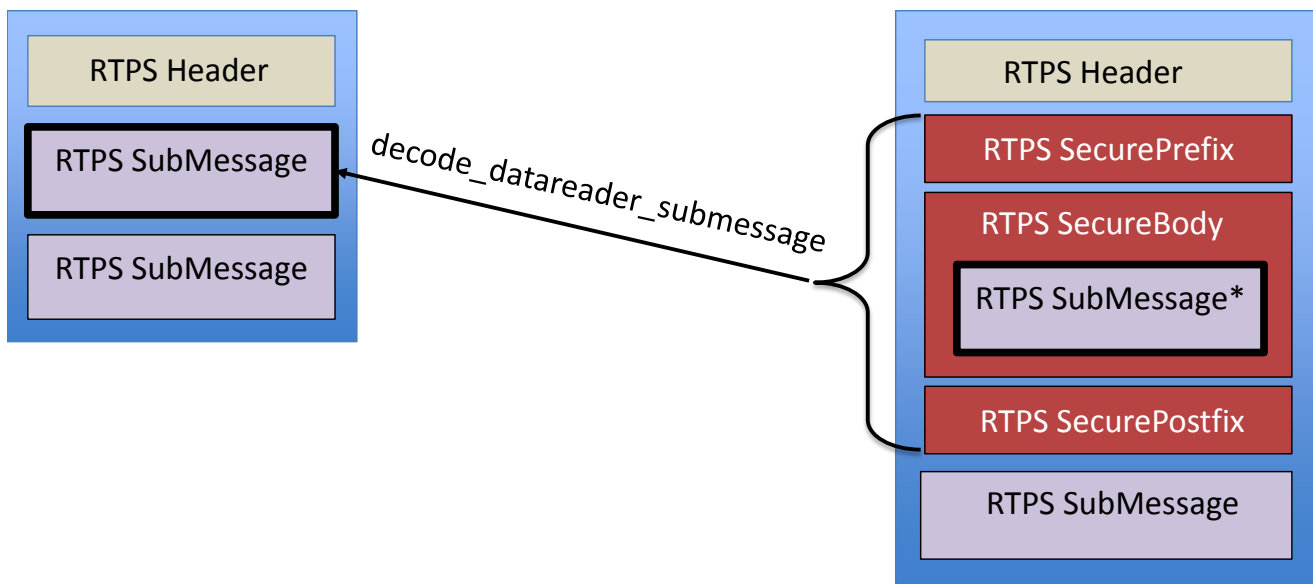
Figure 18 – Effect of decode_datawriter_submessage within an RTPS message

Parameter **plain_rtps_submessage**: The output containing the RTPS submessage created by a DataReader. This submessage will be one of following kinds: `AckNack`, `NackFrag`.

Parameter **encoded_rtps_submessage**: The input containing the RTPS `SecurePrefixSubMsg` and any associated submessages following (for example, `SecureBodySubMsg` and `SecurePostfixSubMsg`), which was created by a call to `encode_datareader_submessage`.

Parameter **receiving_datawriter_crypto**: The `DatawriterCryptoHandle` returned by the preceding call `to preprocess_secure_subessage` performed on the received `SecurePrefixSubMsg`. It shall contain the `DatawriterCryptoHandle` corresponding to the DataWriter that is receiving the RTPS Submessage.

Parameter **sending_datareader_crypto**: The `DatareaderCryptoHandle` returned by the preceding call to `preprocess_secure_submessage` performed on the received `SecurePrefixSubMsg`. It shall contain the `DatareaderCryptoHandle` corresponding to the DataReader that is sending the RTPS Submessage.

8.5.1.9.9 Operation: decode_serialized_payload

This operation shall be called by the DDS implementation as a result of a DataReader receiving a `Data` or `DataFrag` submessage containing a `CryptoContent` RTPS submessage element (instead of the normal `SerializedPayload`).

The operation shall receive in the ***inline_qos*** parameter the InlineQos RTPS `SubmessageElement` that appeared in the RTPS `Data` submessage that carried the `SerializedPayload`.

The DDS implementation shall substitute the `CryptoContent` submessage element within the received submessages with the `SerializedPayload` produced by this operation.

The implementation of `decode_serialized_payload` shall undo the cryptographic transformation of the `SerializedPayload` that was performed by the corresponding call to `encode_serialized_payload` on the DataWriter side. The DDS implementation shall use the available information on the remote DataWriter that wrote the message and the receiving DataReader to locate the corresponding `DatawriterCryptoHandle` and `DatareaderCryptoHandle` and pass them as parameters to the operation. In addition, it shall use the `CryptoTransformIdentifier` present in the `CryptoContent` to verify that the correct key us available and obtain any additional data needed to decode the `CryptoContent`.
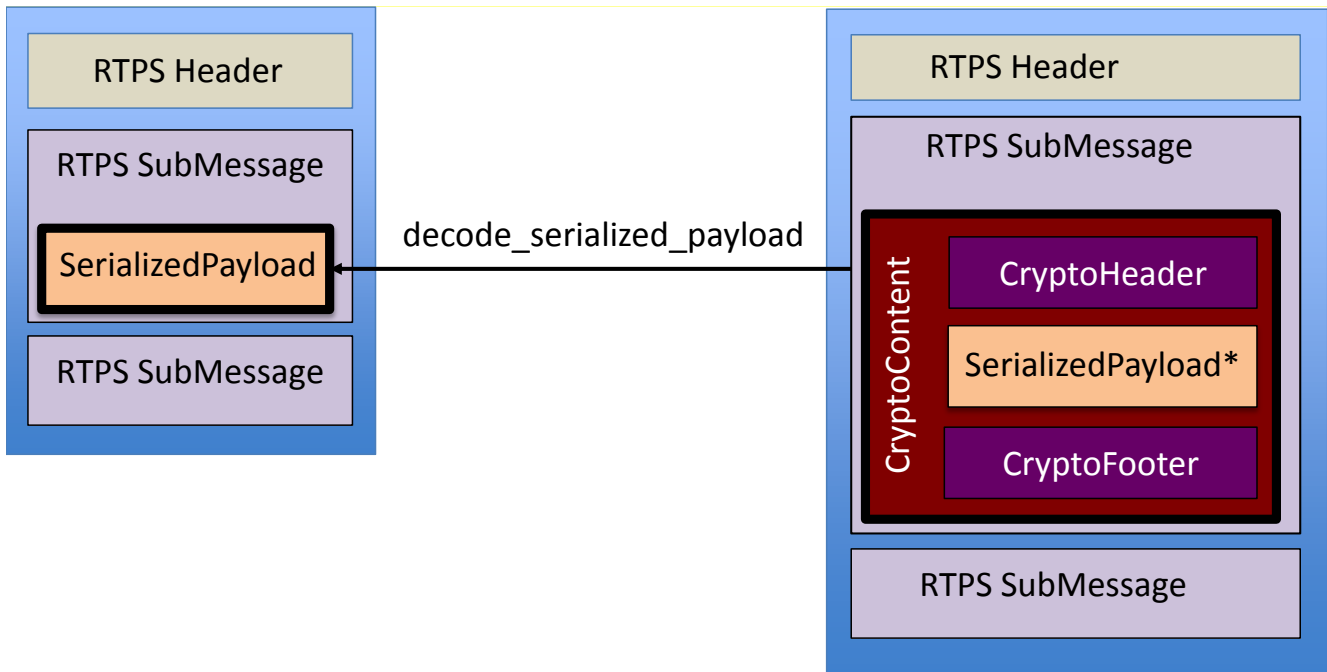
Figure 19 – Effect of decode_serialized_payload within an RTPS message

If an error occurs, this method shall return `false`.

Parameter **plain_buffer**: The output containing the `SerializedPayload` RTPS submessage element, which shall be used to replace the input plain_buffer.

Parameter **encoded_buffer**: The input containing the `CryptoContent` RTPS submessage element.

Parameter **receiving_reader_crypto**: The `DatareaderCryptoHandle` returned by a previous call to `register_local_datareader` for the `DataReader` that received the Submessage containing the `CryptoContent`.

Parameter **sending_datawriter_crypto**: The `DatawriterCryptoHandle` returned by a previous call to `register_matched_remote_datawriter` for the `DataWriter` that wrote the `CryptoContent`.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.6   The Logging Plugin

The Logging Control Plugin API defines the types and operations necessary to support logging of security events for a DDS `DomainParticipant`.

8.6.1 Background (Non-Normative)

The `Logging` plugin provides the capability to log all security events, including expected behavior and all security violations or errors. The goal is to create security logs that can be used to support audits. The rest of the security plugins will use the logging API to log events.

The `Logging` plugin will add an ID to the log message that uniquely specifies the `DomainParticipant`. It will also add a time-stamp to each log message.

The `Logging` API has two options for collecting log data. The first is to log all events to a local file for collection and storage. The second is to distribute log events securely over DDS.

8.6.2 Logging Plugin Model

The logging model is shown in the figure below.
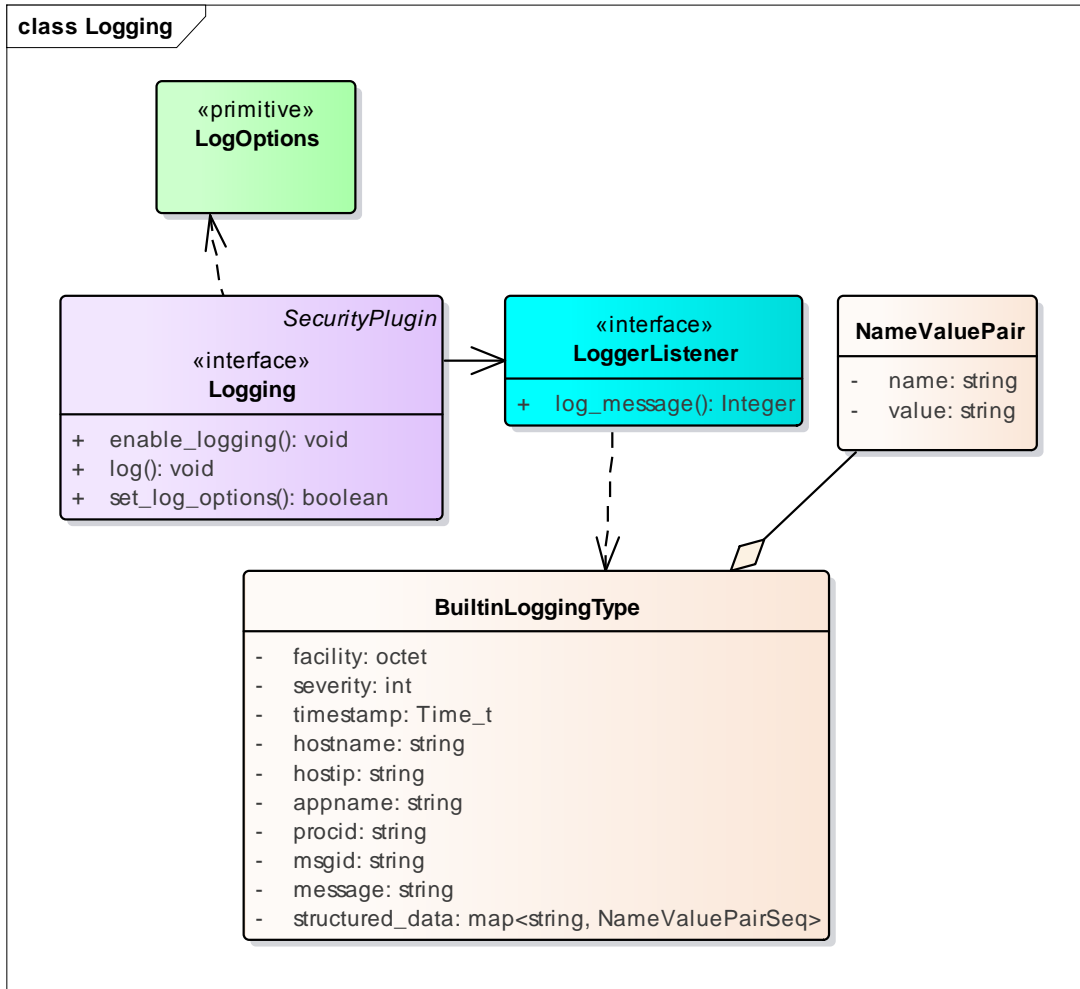
Figure 20 – Logging Plugin Model

8.6.2.1 LogOptions

The `LogOptions` let the user control the ***log level*** and where to log. The options must be set before logging starts and may not be changed at run-time after logging has commenced. This is to ensure that an attacker cannot temporarily suspend logging while they violate security rules, and then start it up again.

The options specify if the messages should be logged to a file and, if so, the file name. The `LogOptions` also specify whether the log messages should be distributed to remote services or only kept locally.

Table 39 – LogOptions values

LogOptions	
Attributes	
`log_level`	`Long`
`log_file`	`String`
`distribute`	`Boolean`

8.6.2.1.1　Attribute: log_level

Specifies what level of log messages will be logged. Messages at or below the ***log_level*** are logged. The levels are as follows, from low to high:

- FATAL_LEVEL – security error causing a shutdown or failure of the Domain Participant

- SEVERE_LEVEL – major security error or fault

- ERROR_LEVEL – minor security error or fault

- WARNING_LEVEL – undesirable or unexpected behavior

- NOTICE_LEVEL – important security event

- INFO_LEVEL – interesting security event

- DEBUG_LEVEL – detailed information on the flow of the security events

- TRACE_LEVEL – even more detailed information

8.6.2.1.2　Attribute: log_file

Specifies the full path to a local file for logging events. If the file already exists, the logger will append log messages to the file. If it is NULL, then the logger will not log messages to a file.

8.6.2.1.3　Attribute: distribute

Specifies whether the log events should be distributed over DDS. If it is TRUE, each log message at or above the `log_level` is published as a DDS `Topic`.

8.6.2.2　Logging

Table 40 – Logging Interface

Logging			
No Attributes			
Operations			
`set_log_options`			`Boolean`
	`options`		`LogOptions`
	`out: exception`		`SecurityException`

log		void
	log_level	long
	message	String
	category	String
	out:exception	SecurityException
enable_logging		void
	out: exception	SecurityException
set_listener		Boolean
	listener	LoggerListener
	out: exception	SecurityException

8.6.2.2.1　Operation: set_log_options

Sets the options for the logger. This must be called before `enable_logging`; it is an error to set the options after logging has been enabled.

If the options are not successfully set, then the method shall return `false`.

Parameter options: the `LogOptions` object with the required options.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.6.2.2.2　Operation: log

Log a message. The logger shall log the message if its `log_level` is at or above the level set in the `LogOptions`. The Logger shall add to the message the RTPS GUID of the `DomainParticipant` whose operations are being logged.

The `Logger` shall populate the *facility*, *severity*, and *timestamp*, fields. The `Logger` may populate the *hostname*, *hostip*, *appname*, *procid* fields as appropriate. The `Logger` shall add an entry to the *structured_data* field with the key "DDS". This `NameValuePair` sequence shall include the following name-value pairs:

Table 41 – Logger structured_data entries

Name	Value
guid	RTPS GUID of the DDS entity that triggered the log message
domain_id	Domain Id of the DomainParticipant that triggered the log message
plugin_class	Identifier of the type of security plugin: Authentication, AccessControl, Cryptographic, etc.
plugin_method	Security plugin method name that triggered the log message

The Logger may add more entries as appropriate for the error condition.

Parameter log_level: The level of the log message. It must correspond to one of the levels defined in 8.6.2.1.1.

Parameter message: The log message.

Parameter category: A category for the log message. This can be used to specify which security plugin generated the message.

Parameter exception: A `SecurityException` object that will return an exception if there is an error with logging.

8.6.2.2.3   Operation: enable_logging

Enables logging. After this method is called, any call to log shall log the messages according to the options. After this method is called, the options may not be modified. This is to ensure that the logger cannot be temporarily suspended to cover up an attack.

If the options are not successfully set, then the method shall return `false`.

Parameter options: the `LogOptions` object with the required options.

Parameter exception: A `SecurityException` object, which provides details in case this operation returns `false`.

8.6.2.2.4   Operation: set_listener

Sets the `LoggerListener` that the `Logger` plugin will use to notify the application of log events. If an error occurs, this method shall return false and fill the `SecurityException`.

Parameter listener: A `LoggerListener` object to be attached to the `Logger` object. If this argument is NIL, it indicates that there shall be no listener.

Parameter exception: A `SecurityException` object, which provides details in case the operation returns FALSE.

8.7 Data Tagging

Data tagging is the ability to add a security label or tag to data. This is often used to specify a classification level of the data including information about its releasability. In a DDS context, it could have several uses:

- It can be used for access control – access control would be granted based on the tag.

- It could be used for message prioritization.

- It could not be used by the middleware, and instead used by the application or other service.

8.7.1 Background (Non-Normative)

There are four different approaches to data tagging:

1. `DataWriter` tagging: data received from a certain `DataWriter` has the tag of the `DataWriter`. This solution does not require the tag to be added to each individual sample.
2. Data instance tagging: each instance of the data has a tag. This solution does not require the tag to be added to each individual sample.
3. Individual sample tagging: every DDS sample has its own tag attached.
4. Per-field sample tagging: very complex management of the tags.

This specification supports `DataWriter` tagging. This was considered the best choice as it meets the majority of uses cases. It fits into the DDS paradigm, as the metadata for all samples from a `DataWriter` is the same. It is also the highest performance, as the tag only needs to be exchanged once when the `DataWriter` is discovered, not sent with each sample.

This approach directly supports typical use cases where each application or `DomainParticipant` writes data on a `Topic` with a common set of tags (e.g., all at the same specified security level). For use cases where an application creates data at different classifications, that application can create multiple `DataWriters` with different tags.

8.7.2 DataTagging Model

The `DataWriter` tag will be associated with every sample written by the `DataWriter`. The `DataWriter` DataTag is implemented as an immutable `DataWriterQos`. The `DataWriter` DataTag shall be propagated via in the `PublicationBuiltinTopicData` as part of the DDS discovery protocol.

The `DataReader` DataTag is implemented as an immutable `DataReaderQos`. The `DataReader` DataTag shall be propagated via in the `SubscriptionBuiltinTopicData` as part of the DDS discovery protocol.

8.8 Security Plugins Behavior

In the previous sub clauses, the functionality and APIs of each plugin have been described. This sub clause provides additional information on how the plugins are integrated with the middleware.

8.8.1 Authentication and AccessControl behavior with local DomainParticipant

The figure below illustrates the functionality of the security plugins with regards to a local `DomainParticipant`.

In this sub clause the term *"DDS application"* refers to the application code that calls the DDS API. The term "*DDS middleware*" refers to a DDS Implementation that complies with the DDS Security specification.
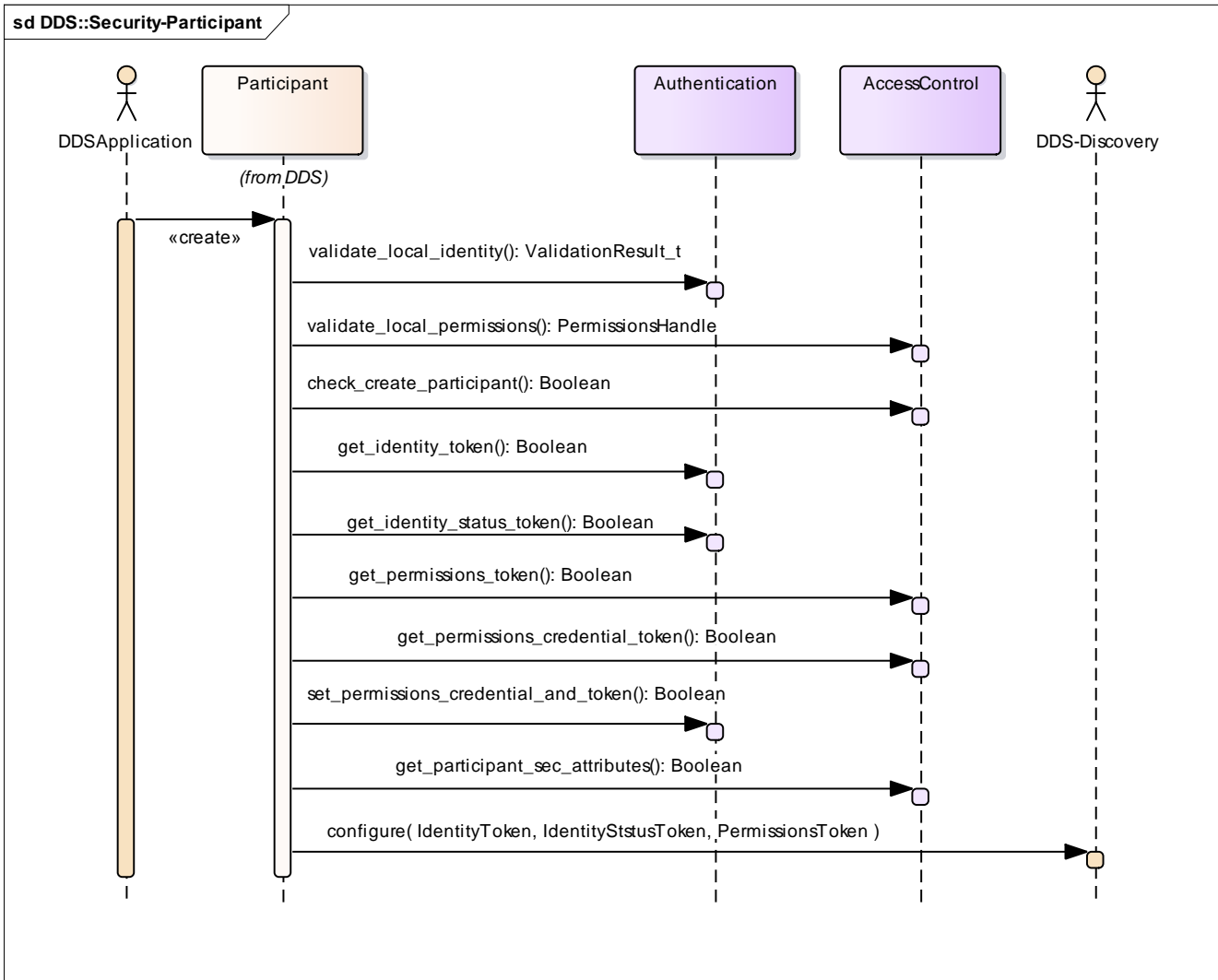
Figure 21 – Authentication and AccessControl sequence diagram with local DomainParticipant

This behavior sequence is triggered when the DDS application initiates the creation of a local `DomainParticipant` by calling the `create_participant` operation on the `DomainParticipantFactory`. The following are mandatory steps that the DDS middleware shall perform prior to creating the `DomainParticipant`. The steps need not occur exactly as described as long as the observable behavior matches the one described below.

The DDS middleware shall validate the identity of the application attempting to create the `DomainParticipant` by calling the `Authentication::validate_local_identity` operation, passing the `domain_id`, the `DomainParticipantQos`, and a *candidate_participant_guid*. The `Authentication` plugin validates the identity of the local `DomainParticipant` and returns an `IdentityHandle` for the holder of the identity (`DomainParticipant`), which will be necessary for interacting with the access control plugin. The `validate_local_identity` operation also returns an adjusted_participant_guid. If the identity is not successfully validated, the DDS middleware shall not create the `DomainParticipant` and

the `create_participant` operation shall return NIL and set the return code to NOT_ALLOWED_BY_SECURITY.

1. The DDS middleware shall validate that the DDS application has the necessary permissions to join DDS domains by calling the `AccessControl::validate_local_permissions` operation. The Access Control plugin shall validate the permissions and issue a signed `PermissionsHandle` for the holder of the identity (`DomainParticipant`). If the permissions are not validated, the `DomainParticipant` shall not be created, the `create_participant` operation shall return NIL and set the return code to NOT_ALLOWED_BY_SECURITY.

2. The DDS middleware shall verify that the DDS application has the necessary permissions to join the specific Domain identified by the `domainId` by calling the operation `AccessControl::check_create_participant`. If this operation returns FALSE, the `DomainParticipant` shall not be created, the `create_participant` operation shall return NIL and set the return code to NOT_ALLOWED_BY_SECURITY.

3. The DDS middleware shall call the `get_identity_token` operation to obtain the `IdentityToken` object corresponding to the received `IdentityHandle`. The `IdentityToken` object shall be placed in the ParticipantBuiltinTopicData sent via discovery, see 7.4.1.3.

4. The DDS middleware shall call the `get_identity_status_token` operation to obtain the `IdentityStatusToken` object corresponding to the received `IdentityHandle`. If the returned `IdentityStatusToken` object is different than `TokenNIL`, it shall be placed in the ***ParticipantBuiltinTopicDataSecure*** sent via secure discovery, see 7.4.1.6.

5. The middleware shall call the `get_permissions_token` operation on the AccessControl plugin to obtain the `PermissionsToken` object corresponding to the received `PermissionsHandle`. The `PermissionsToken` shall be placed in the ParticipantBuiltinTopicData sent via discovery, see 7.4.1.3.

6. The middleware calls `the get_permissions_credential_token` operation on the `AccessControl` plugin, which returns the `PermissionsCredentialToken` object corresponding to the received `PermissionsHandle`. The `PermissionsCredentialToken` object is necessary to configure the `Authentication` plugin.

7. The middleware calls the `set_permissions_credential_and_token` operation on the `Authentication` plugin such that it can be sent during the authentication handshake.

8. The middleware calls the `get_participant_sec_attributes` operation on the `AccessControl` plugin to obtain the `ParticipantSecurityAttributes` such that it knows how to handle remote participants that fail to authenticate.

9. The `DomainParticipant`'s `IdentityToken`, and `PermissionsToken` are used to configure DDS discovery such that they are propagated inside the ***identity_token*** and the ***permissions_token*** members of the ***ParticipantBuiltinTopicData***. The `DomainParticipant`'s `IdentityStatusToken`, is used to configure DDS

discovery such it is propagated inside the ***identity_status_token*** member of the ***ParticipantBuiltinTopicDataSecure***. This operation is internal to the DDS implementation and therefore this API is not specified by the DDS Security specification. It is mentioned here to provide guidance to implementers.

8.8.2 Authentication behavior with discovered DomainParticipant

Depending on the `ParticipantSecurityAttributes` returned by the `AccessControl` operation `get_participant_sec_attributes` the `DomainParticipant` may allow remote `DomainParticipants` that lack the ability to authenticate (e.g., do not implement DDS Security) to match.

8.8.2.1 Behavior when `allow_unauthenticated_participants` is set to TRUE

If the `ParticipantSecurityAttributes` returned by the operation `get_participant_sec_attributes` has the member `allow_unauthenticated_participants` set to TRUE, the `DomainParticipant` shall allow matching remote `DomainParticipant` entities that are not able to authenticate. Specifically:

- Discovered `DomainParticipant` entities that do *not* implement the DDS Security specification or do not contain compatible Security Plugins shall be matched without the `DomainParticipant` attempting to authenticate them and shall be treated as "Unauthenticated" `DomainParticipant` entities.
- Discovered `DomainParticipant` entities that *do* implement the DDS Security specification and declare compatible Security Plugins but fail the Authentication protocol shall be matched and treated as "Unauthenticated" `DomainParticipants` entities.

  For any matched "Unauthenticated" `DomainParticipant` entities, the `DomainParticipant` shall **match only** the regular builtin Endpoints (***ParticipantMessage***, ***DCPSParticipants***, ***DCPSPublications***, ***DCPSSubscriptions***) **and not** the builtin secure Endpoints (see 7.4.5 for the complete list).

  For any matched authenticated `DomainParticipant` entities, the `DomainParticipant` shall match all the builtin endpoints.

8.8.2.2 Behavior when `allow_unauthenticated_participants` is set to FALSE

If the `ParticipantSecurityAttributes` has the member `allow_unauthenticated_participants` set to FALSE, the `DomainParticipant` shall reject remote `DomainParticipant` entities that are not able to authenticate. Specifically:

- Discovered `DomainParticipant` entities that do not implement the DDS Security specification or do not contain compatible Security Plugins shall be rejected without the `DomainParticipant` attempting to authenticate them.
- Discovered `DomainParticipant` entities that do implement the DDS Security specification, declare compatible Security Plugins but fail the Authentication protocol shall be rejected.
- Discovered `DomainParticipant` entities that do implement the DDS Security specification and declare compatible Security Plugins automatically "match" the ***ParticipantStatelessMessage*** builtin endpoints to allow the authentication handshake to proceed.

- Discovered `DomainParticipant` entities that do implement the DDS Security specification, declare compatible Security Plugins, and pass the Authentication protocol successfully shall be matched and the `DomainParticipant` shall also match all the builtin endpoints of the discovered `DomainParticipant`, except for the ***ParticipantStatelessMessage*** builtin endpoints, which were already matched prior to the Authentication protocol.

The figure below illustrates the behavior of the security plugins with regards to a discovered `DomainParticipant` that also implements the DDS Security specification and announces compatible security plugins. The exact operations depend on the plugin implementations. The sequence diagram shown below is just indicative of one possible sequence of events and matches what the builtin DDS:Auth:PKI-DH plugin (see 9.3.3) does.
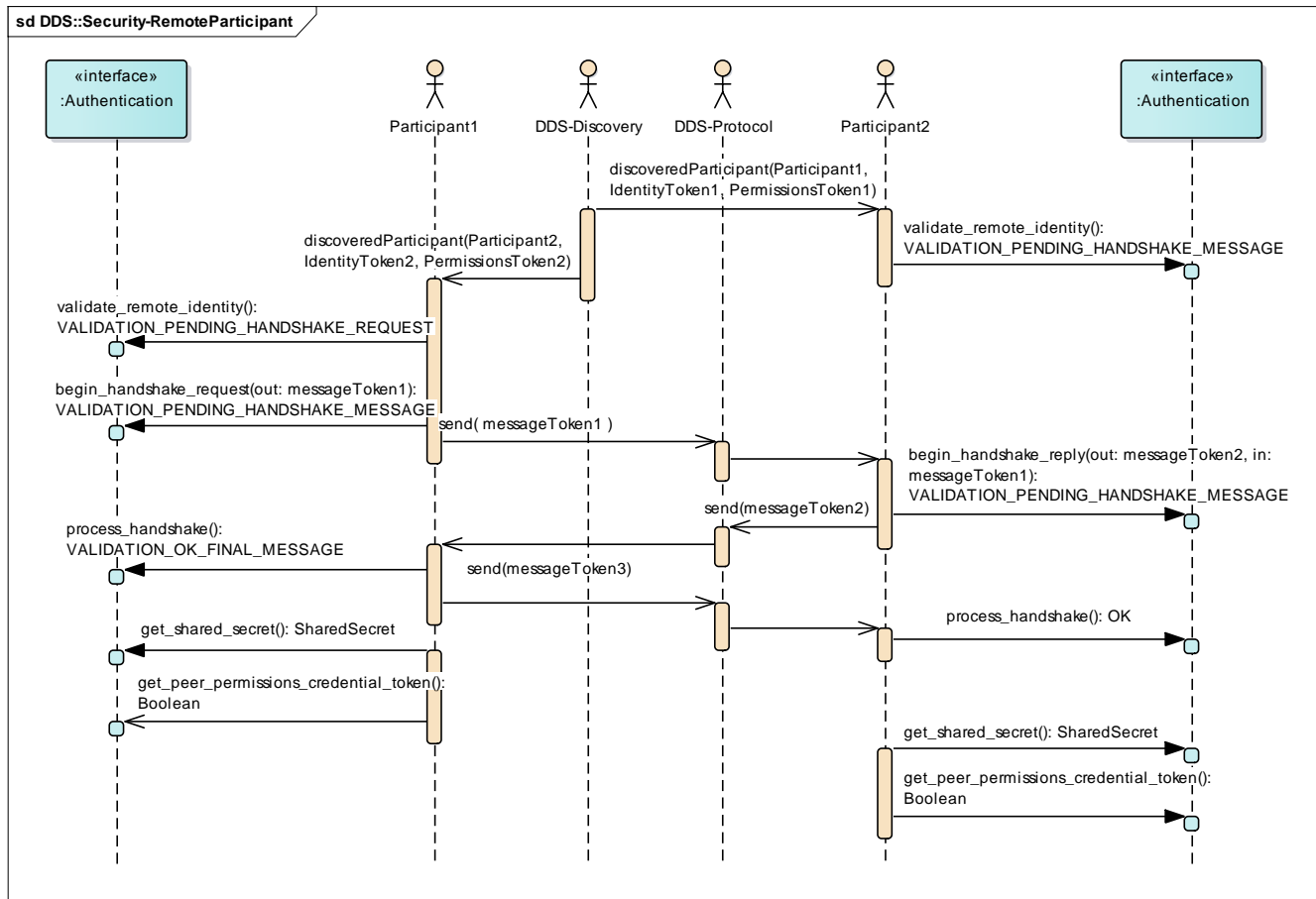
Figure 22 – Authentication sequence diagram with discovered DomainParticipant

1. Participant2 discovers Participant1via the discovery protocol. The `BuiltinParticipantTopicData` contains the `IdentityToken` and `PermissionsToken` of Participant1.

2. Participant2 calls the `validate_remote_identity` operation to validate the identity of Participant1 passing the local `IdentityHandle` of Participant2 and the remote `IdentityToken` and `GUID_t` of Participant1 received via discovery and obtains an `IdentityHandle` for Participant1, needed for further operations involving Participant1. The operation returns VALIDATION_PENDING_HANDSHAKE_MESSAGE indicating that further handshake messages are needed to complete the validation and that Participant2

should wait for a `HandshakeMessageToken` to be received from Participant1. Participant2 waits for this message.

3. Participant1 discovers Participant2 via the DDS discovery protocol. The `BuiltinParticipantTopicData` contains the `IdentityToken` and `PermissionsToken` of Participant2.

4. Participant1 calls the operation `validate_remote_identity` to validate the identity of Participant2 passing the `IdentityToken` and `PermissionsToken` of Participant2 received via discovery and obtains an `IdentityHandle` for Participant2, needed for further operations involving Participant2. The operation returns VALIDATION_PENDING_HANDSHAKE_REQUEST indicating further handshake messages are needed and Participant1 should initiate the handshake.

5. Participant1 calls `begin_handshake_request` to begin the requested handshake. The operation outputs a `HandshakeHandle` and a `HandshakeMessageToken` (messageToken1). The operation returns VALIDATION_PENDING_HANDSHAKE_MESSAGE indicating authentication is not complete and the returned messageToken1 needs to be sent to Participant2 and a reply should be expected.

6. Participant1 sends the `HandshakeMessageToken` (messageToken1) to Participant2 using the ***BuiltinParticipantMessageWriter***.

7. Participant2 receives the `HandshakeMessageToken` (messageToken1) on the ***BuiltinParticipantMessageReader***. Participant2 determines the message originated from a remote `DomainParticipant` (Participant1) for which it had already called `validate_remote_identity` where the function had returned VALIDATION_PENDING_HANDSHAKE_REPLY.

8. Participant2 calls `begin_handshake_reply` passing the received `HandshakeMessageToken` (messageToken1). The `Authentication` plugin processes the `HandshakeMessageToken` (messageToken1) and outputs a `HandshakeMessageToken` (messageToken2) in response and a `HandshakeHandle`. The operation `begin_handshake_reply` returns VALIDATION_PENDING_HANDSHAKE_MESSAGE, indicating authentication is not complete and an additional message needs to be received.

9. Participant2 sends the `HandshakeMessageToken` (messageToken2) back to Participant1 using the ***BuiltinParticipantMessageWriter***.

10. Participant1 receives the `HandshakeMessageToken` (messageToken2) on the ***BuiltinParticipantMessageReader***. Participant1 determines this message originated from a remote `DomainParticipant` (Participant2) for which it had already called `validate_remote_identity` where the function had returned VALIDATION_PENDING_HANDSHAKE_REQUEST.

11. Participant1 calls `process_handshake` passing the received `HandshakeMessageToken` (messageToken2). The Authentication plugin processes messageToken2, verifies it is a valid reply to the messageToken1 it had sent and outputs the `HandshakeMessageToken` messageToken3 in response. The `process_handshake`

operation returns VALIDATION_OK_FINAL_MESSAGE, indicating authentication is complete but the returned `HandshakeMessageToken` (messageToken3) must be sent to Participant2.

12. Participant1 sends the `HandshakeMessageToken` (messageToken3) to Participant2 using the ***BuiltinParticipantMessageWriter***.

13. Participant2 receives the `HandshakeMessageToken` (messageToken3) on the ***BuiltinParticipantMessageReader***. Participant2 determines this message originated from a remote `DomainParticipant` (Participant1) for which it had already called the operation `begin_handshake_reply` where the call had returned VALIDATION_PENDING_HANDSHAKE_MESSAGE.

14. Participant2 calls the `process_handshake` operation, passing the received `HandshakeMessageToken` (messageToken3). The Authentication plugin processes the messageToken2, verifies it is a valid reply to the messageToken2 it had sent and returns OK, indicating authentication is complete and no more messages need to be sent or received.

15. Participant1, having completed the authentication of Participant2, calls the operation `get_shared_secret` to retrieve the `SharedSecret`, which is used with the other Plugins to create `Tokens` to exchange with Participant2.

16. Participant1, having completed the authentication of Participant2, calls the operation `get_authenticated_peer_credential_token` to retrieve the `AuthenticatedPeerCredentialToken` associated with Participant2, which is used with the `AccessControl` plugin to determine the permissions that Participant1 will grant to Participant2.

17. Participant2, having completed the authentication of Participant1, calls the operation `get_shared_secret` to retrieve the `SharedSecret`, which is used with the other Plugins to create `Tokens` to exchange with Participant1.

18. Participant2, having completed the Authentication of Participant1, calls the operation `get_authenticated_peer_credential_token` to retrieve the `AuthenticatedPeerCredentialToken` associated with Participant2 which is used with the `AccessControl` plugins to determine the permissions that Participant2 will grant to Participant1.

8.8.3 DDS Entities impacted by the AccessControl operations

There are six types of `DDS Entities`: `DomainParticipant`, `Topic`, `Publisher`, `Subscriber`, `DataReader`, and `DataWriter`. All these except the `DomainParticipant` are defined as the `DDS Domain Entities` (subclause 2.2.2.1.2 of DDS [1]).

The `Domain Entities` created by a `DomainParticipant` can be grouped into four categories:

- DDS-RTPS Protocol [2] Builtin Entities. These are domain entities used to read and write the four builtin Topics: ***DCPSParticipants***, ***DCPSTopics***, ***DCPSPublications***, ***DCPSSubscriptions***.

- Builtin Secure Entities. These are the `Domain Entities` related to the Builtin Secure Endpoints defined in Section 7.4.5. These `Entities` are used to read and write the four builtin secure topics: ***DCPSPublicationsSecure***, ***DCPSSubscriptionsSecure***, ***DCPSParticipantMessageSecure***, and ***DCPSParticipantVolatileMessageSecure***.
- Other builtin `Entities` defined by the DDS-Security specification not included in the "Builtin Secure Endpoints". These are the ***BuiltinParticipantStatelessMessageWriter*** and the ***BuiltinParticipantStatelessMessageReader***.
- Application-defined `Entities`. These are any non-builtin `Domain Entities`.

The `AccessControl` plugin shall impact only the Builtin Secure Entities and the application-defined Entities. It shall not impact the builtin entities defined by the DDS-RTPS Protocol specification nor the ***BuiltinParticipantStatelessMessageWriter*** or the ***BuiltinParticipantStatelessMessageReader***.

`AccessControl` plugin operations can be grouped into 5 groups:

1. Group1. Operations related to `DomainParticipant`. These are: validate_local_permissions, validate_remote_permissions, check_create_participant, get_permissions_token, get_permissions_credential_token, set_listener, return_permissions_token, return_permissions_credential_token, get_participant_sec_attributes, return_participant_sec_attributes.
2. Group2. Operations related to the creation of local `Domain Entities`. These are: check_create_topic, check_create_datawriter, check_create_datareader, get_datawriter_sec_attributes, get_datareader_sec_attributes, return_datawriter_sec_attributes, return_datareader_sec_attributes.
3. Group3. Operations related to write activities of local `Domain Entities`. These are: check_local_datawriter_register_instance and check_local_datawriter_dispose_instance.
4. Group4. Operations related to discovery and match of remote `Domain Entities`. These are: check_remote_topic, check_remote_datawriter, check_remote_datareader, check_local_datawriter_match, and check_local_datareader_match.
5. Group5. Operations related to the write activities of remote `Domain Entities`. These are: check_remote_datawriter_register_instance and check_remote_datawriter_dispose_instance.

Table 42 below summarizes the DDS `Entities` affected by each operation group.

Table 42 – Impact of Access Control Operations to the DDS Builtin and Application-defined Entities

Entity Category	*Entity*	*Impact by AccessControl operation in group*				
		Group1	*Group2*	*Group3*	*Group4*	*Group5*
DomainParticipant	All created	Yes	No	No	No	No
DDS-RTPS Protocol Builtin Entities	See RTPS Protocol specification [2]	Yes, indirectly	No	No	No	No

Builtin Secure Entities	SEDPbuiltinPublicationsSecureWriter SEDPbuiltinPublicationsSecureReader SEDPbuiltinSubscriptionsSecureWriter SEDPbuiltinSubscriptionsSecureReader BuiltinParticipantMessageSecureWriter BuiltinParticipantMessageSecureReader BuiltinParticipantVolatileMessageSecureWriter BuiltinParticipantVolatileMessageSecureReader	Yes, indirectly	Only get_datawriter_sec_attributes and get_datareader_sec_attributes	No	No	No
Other builtin Entities defined by DDS-Security	BuiltinParticipantStatelessMessageWriter BuiltinParticipantStatelessMessageReader	Yes, indirectly	No	No	No	No
Application-defined Domain Entities	Publisher, Subscriber	Yes, indirectly	Yes, indirectly	No	Yes, indirectly	No
	Topic, DataWriter, DataReader	Yes, indirectly	Yes	Yes	Yes	Yes

The `DomainParticipant` entities are only impacted by `AccessControl` plugin operations in Group1. The `DomainParticipant` is not created unless allowed by the `AccessControl` plugin. Also the matching of a remote `DomainParticipant` must be allowed by the `AccessControl` plugin. The full interaction is described in subclauses 8.8.1 and 8.8.6.

The DDS-RTPS Builtin Entities are impacted indirectly by `AccessControl` plugin operations in Group1 in the sense that if the sense that the creation of the `Entities` is dependent on the successful creation of the local `DomainParticipant` which is controlled by the Group1 operations. Likewise the match of the remote entities is dependent on the successful match of a remote `DomainParticipant`, which is also controlled by the Group1 operations.

The DDS-RTPS Builtin Entities shall not be impacted by any of the operations in Group2, Group3, Group4, or Group5.

The Secure Builtin Entities are impacted indirectly by `AccessControl` plugin operations in Group1 in the same way as the DDS-RTPS Builtin Entities.

The Secure Builtin Entities are impacted only by the `get_datawriter_sec_attributes` and `get_datareader_sec_attributes` operations in Group2. They shall not be impacted by any other Group2 operations. This means that the Secure Builtin Entities shall be created unconditionally when the DomainParticipant is created. During the creation process of DataWriter entities the `get_datawriter_sec_attributes` shall be called and likewise during the creation process of DataReader entities the `get_datareader_sec_attributes` shall be called. The purpose of calling these `get_xxx_sec_attributes` operations is to obtain the information necessary to call the Cryptographic plugin operations on these endpoints.

The ***BuiltinParticipantStatelessMessageWriter*** and ***BuiltinParticipantStatelessMessageReader*** are only indirectly impacted by the Group2 operations in that they are tied to the successful creation of the `DomainParticipant`. They are not impacted by the successful match of remote entities not any other `AccessControl` plugin operations in any Group. DDS Secure implementations shall create these endpoints unconditionally for all created `DomainParticipant`. Being stateless these endpoints are not "matched" to remote endpoints in the sense of being aware and maintaining the state and presence of the remote endpoints. Nevertheless they are able to send exchange information in a stateless, best-efforts manner.

The Application-defined `Publisher` and `Subscriber Entities` are impacted indirectly by `AccessControl` plugin operations in Group1 only by the fact that they depend on the successful creation of the `DomainParticipant`. They are impacted indirectly by operations in Group2 by the fact that the PartitionQos settings of the `Publisher` (or `Subscriber`) may cause the `AccessControl` plugin to prevent the creation of `DataWriter` (or `DataReader`) entities belonging to them. Likewise they are impacted indirectly by operations in Group4 in that the `PartitionQos` settings of the remote `Publisher` (or `Subscriber`) may cause the `AccessControl` plugin to prevent matching of `remote DataWriter` (or `DataReader`) entities. They are not impacted by operations in Group3 or Group5.

The Application-defined Topic, `DataWriter` and `DataReader` entities are impacted indirectly by `AccessControl` plugin operations in Group1 the same way the The DDS-RTPS Builtin Entities are. These `Entities` are impacted by the `AccessControl` plugin operations in Group2, Group3, Group4, and Group5. This is described in subclauses 8.8.5 and 8.8.7.

8.8.4 AccessControl behavior with local participant creation

The functionality of the `AccesControl` plugin with regards to the creation of local DDS `DomainParticipant` entities was illustrated in Figure 21 and described in 8.8.1. Subclause 8.8.1 covered `Authentication` and `AccessControl` plugin behavior simultanepusly because these two plugins interact with each other.

8.8.5 AccessControl behavior with local domain entity creation

The figure below illustrates the functionality of the security plugins with regards to the creation of local DDS domain entities: `Topic`, `DataWriter`, and `DataReader` entities.
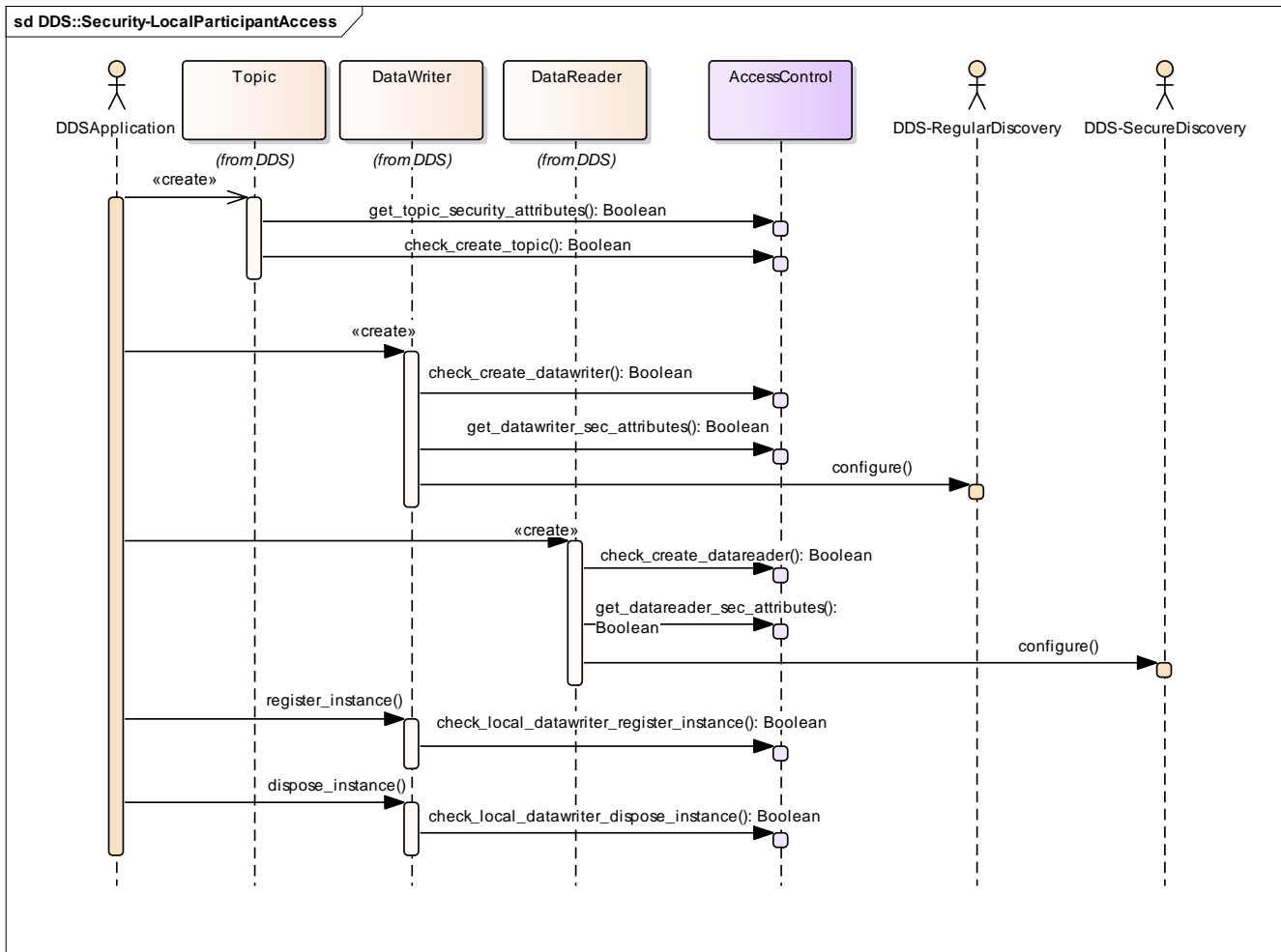
Figure 23 – AccessControl sequence diagram with local entities

1. The DDS application initiates the creation of a new `Topic` for the `DomainParticipant`.

2. The middleware calls `AccessControl::get_topic_sec_attributes` to obtain the `TopicSecurityAttributes` for the `Topic` being created.

3. The middleware verifies the `DomainParticipant` is allowed to create a Topic with name `topicName`. Operation `AccessControl::check_create_topic()` is called for this verification. If the verification fails, the Topic object is not created.

4. The DDS application initiates the creation of a local `DataWriter`.

5. The middleware verifies that the `DataWriter` has the right permissions to publish on Topic `topicName`. Operation `AccessControl::check_create_datawriter()` is called for this verification. As an optional behavior, `check_create_datawriter ()` can also verify if the `DataWriter` is allowed to tag data with `dataTag`. If the verification doesn't succeed, the `DataWriter` is not created. As an optional behavior, `check_create_datawriter()` can also check the QoS associated with the `DataWriter` and grant permissions taking that into consideration.

6. The middleware calls `AccessControl::get_datawriter_sec_attributes` to obtain the `EndpointSecurityAttributes` for the created `DataWriter`.

7. This sequence diagram illustrates the situation where the `TopicSecurityAttributes` for the created `DataWriter` has the **is_discovery_protected** attribute set to FALSE. In this situation the middleware configures Discovery to use regular (not secure) publications discovery endpoint (**DCPSPublications**) to propagate the `PublicationBuiltinTopicData` for the created `DataWriter`.

8. The DDS application initiates the creation of a local `DataReader`.

9. The middleware verifies that the `DataReader` has the right permissions to subscribe on Topic `topicName`. Operation `AccessControl::check_create_datareader()` is called for this verification. As an optional behavior, `check_create_datareader()` can also verify if the `DataReader` is allowed to receive data tagged with `dataTag`. If the verification doesn't succeed, the `DataReader` is not created. As an optional behavior `check_create_datareader()` can also check the QoS associated with the `DataReader` and grant permissions taking that into consideration.

10. The middleware calls the operation `AccessControl::get_datareader_sec_attributes` to obtain the `EndpointSecurityAttributes` for the created `DataReader` entity.

11. This sequence diagram illustrates the situation where the `TopicSecurityAttributes` for the topic (a different topic than in the earlier steps) has the **is_discovery_protected** attribute set to TRUE. In this situation the middleware configures Discovery to use the secure subscriptions discovery endpoint (**DCPSSecureSubscriptions**) to propagate the `SubscriptionBuiltinTopicData` for the created `DataReader`.

12. The DDS application initiates the registration of a data instance on the `DataWriter`.

13. The middleware verifies that the `DataWriter` has the right permissions to register the instance. The operation `AccessControl::check_local_datawriter_register_instance()` is called for this verification. If the verification doesn't succeed, the instance is not registered.

14. The DDS application initiates the disposal of an instance of the `DataWriter`.

15. The middleware verifies that the `DataWriter` has the right permissions to dispose the instance. The operation `AccessControl::check_local_datawriter_dispose_instance()` is called for this verification. If the verification doesn't succeed, the instance is not disposed.

8.8.6 AccessControl behavior with remote participant discovery

If the `ParticipantSecurityAttributes` object returned by the `AccessControl` operation `get_participant_sec_attributes` has the **allow_unauthenticated_participants** attribute set to TRUE, the `DomainParticipant` may discover DomainParticipants that cannot be authenticated because they either lack support for the authentication protocol or they fail the authentication protocol. These "Unauthenticated" `DomainParticipant` entities shall be matched and considered "Unauthenticated" DomainParticipant entities. Local `DomainParticipant` will not perform any

further participant AccessControl validation with unauthenticated participants (i.e., `validate_remote_permissions` and `check_remote_participant` will not be called).

If the `DomainParticipant` discovers a DomainParticipant entity that it can authenticate successfully, and *is_access_protected* is TRUE, then it shall validate with the AccessControl plugin that it has the permissions necessary to join the DDS domain. This is done by successfully calling to `get_authenticated_peer_credential_token` on the `Authentication` plugin, then to `validate_remote_permissions` and `check_remote_participant` in the `AccessControl` plugin:

- If the validation succeeds, the discovered DomainParticipant shall be considered "Authenticated" and all the builtin Topics automatically matched.
- If the validation fails, the discovered DomainParticipant shall be considered ignored and all the builtin Topics should not be matched.

If the `DomainParticipant` discovers a `DomainParticipant` entity that it can authenticate successfully, and *is_access_protected* is FALSE, then validation will succeed with no access control checking. In this case, only `get_authenticated_peer_credential_token` and `validate_remote_permissions` are called, and a HandleNIL return will not impact the validation result.

The figure below illustrates the functionality of the security plugins with regards to the discovery of remote `DomainParticipant` entity that has been successfully authenticated by the Authentication plugin.
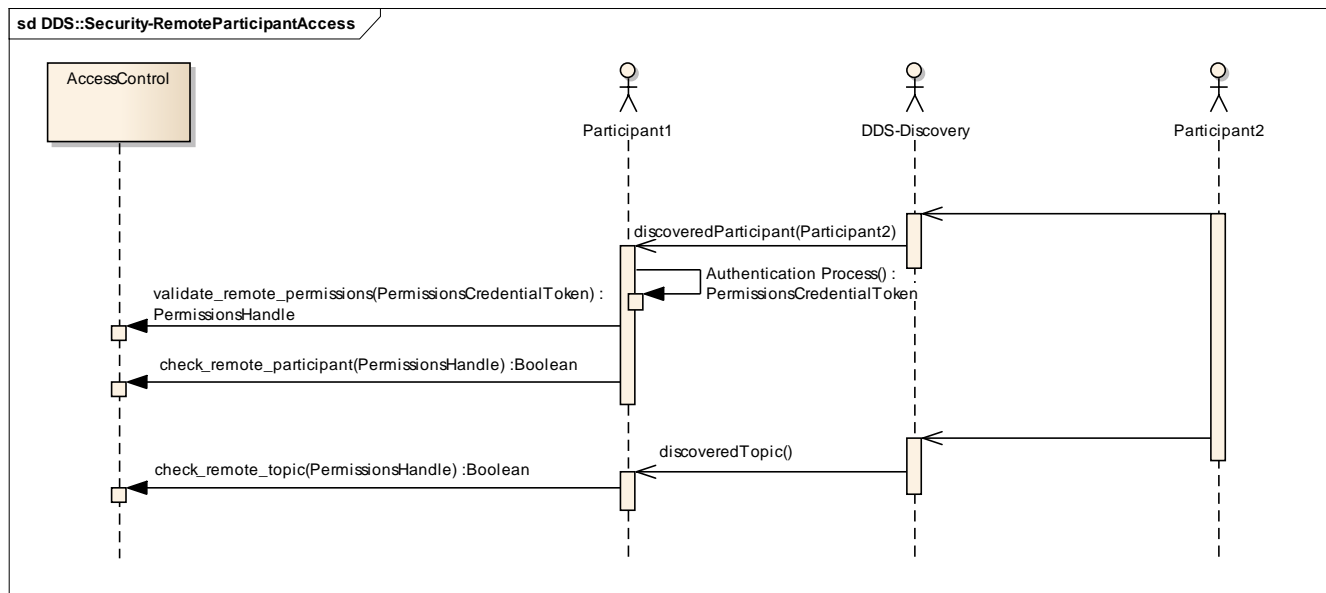
Figure 24 – AccessControl sequence diagram with discovered DomainParticipant

1. The `DomainParticipant` Participant1 discovers the `DomainParticipant` (Participant2) via the discovery protocol and successfully authenticates Participant2 and obtains the `AuthenticatedPeerCredentialToken` as described in 8.8.2.

2. Participant1 calls the operation `validate_remote_permissions` to validate the permissions of Participant2, passing the `PermissionsToken` obtained via discovery from Participant2 and the `AuthenticatedPeerCredentialToken` returned by the

operation `get_authenticated_peer_credential_token` on the `Authentication` plugin. The operation `validate_remote_permissions` returns a `PermissionsHandle`, which the middleware will use whenever an access control decision must be made for the remote `DomainParticipant`.

3. Participant1 calls the operation `check_remote_participant` to verify the remote `DomainParticipant` (Participant2) is allowed to join the DDS domain with the specified `domainId`, passing the `PermissionsHandle` returned by the `validate_remote_permissions` operation. If the verification fails, the remote `DomainParticipant` is ignored and all the endpoints corresponding to the builtin Topics are unmatched.

4. Participant1 discovers that `DomainParticipant` (Participant2) has created a new DDS `Topic`.

5. Participant1 verifies that the remote `DomainParticipant` (Participant2) has the permissions needed to create a DDS `Topic` with name `topicName`. The operation `check_remote_topic` is called for this verification. If the verification fails, the discovered `Topic` is ignored.

8.8.7 AccessControl behavior with remote domain entity discovery

This sub clause describes the functionality of the AccessControl plugin relative to the discovery of remote domain entities, that is, `Topic`, `DataWriter`, and `DataReader` entities.

If the `ParticipantSecurityAttributes` object returned by the `AccessControl` operation `get_participant_sec_attributes` has the `is_access_protected` attribute set to FALSE, the `DomainParticipant` may have matched a remote "Unauthenticated" `DomainParticipant`, i.e., a `DomainParticipant` that has not authenticated successfully and may therefore discover endpoints via the regular (non-secure) discovery endpoints from an "Unauthenticated" `DomainParticipant`.

8.8.7.1 AccessControl behavior with discovered endpoints from "Unauthenticated" DomainParticipant

If the `DomainParticipant` discovers endpoints from an "Unauthenticated" `DomainParticipant` it shall:

- Reject (do not try to match) local endpoints for which the related `TopicSecurityAttributes` have the attribute *is_read_protected* or *is_write_protected* set to TRUE.
- Proceed to try matching without checking with the `AccessControl` plugin the local `DataWriter` endpoints for which the related `TopicSecurityAttributes` object returned by the operation `get_topic_sec_attributes` have the attribute *is_read_protected* set to FALSE.
- Proceed to try matching without checking with the `AccessControl` plugin the local `DataReader` endpoints for which the related `TopicSecurityAttributes` object returned by the operation `get_topic_sec_attributes` have the attribute *is_write_protected* set to FALSE.

8.8.7.2 AccessControl behavior with discovered endpoints from "Authenticated" DomainParticipant

If the `DomainParticipant` discovers endpoints from an "authenticated" `DomainParticipant` it shall:

- Perform the `AccessControl` checks for discovered endpoints that would match local `DataWriters` for whom the **is_read_protected** attribute is set to TRUE, and only proceed to try matching the discovered endpoints for whom the access control checks succeed.
- Perform the `AccessControl` checks for discovered endpoints that would match local `DataReader` for whom the **is_write_protected** attribute is set to TRUE, and only proceed to try matching the discovered endpoints for whom the access control checks succeed.
- Proceed to try matching without checking with the `AccessControl` plugin the local `DataWriters` for whom the related `TopicSecurityAttributes` object returned by the operation `get_topic_sec_attributes` has the **is_read_protected** attribute set to FALSE.
- Proceed to try matching without checking with the `AccessControl` plugin the local `DataReaders` for whom the related `TopicSecurityAttributes` object returned by the operation `get_topic_sec_attributes` has the **is_write_protected** attribute set to FALSE.

The figure below illustrates the behavior relative to discovered endpoints coming from an "Authenticated" `DomainParticipant` that would match local endpoints for which the **is_read_protected** and **is_write_protected** attributes are set to FALSE.
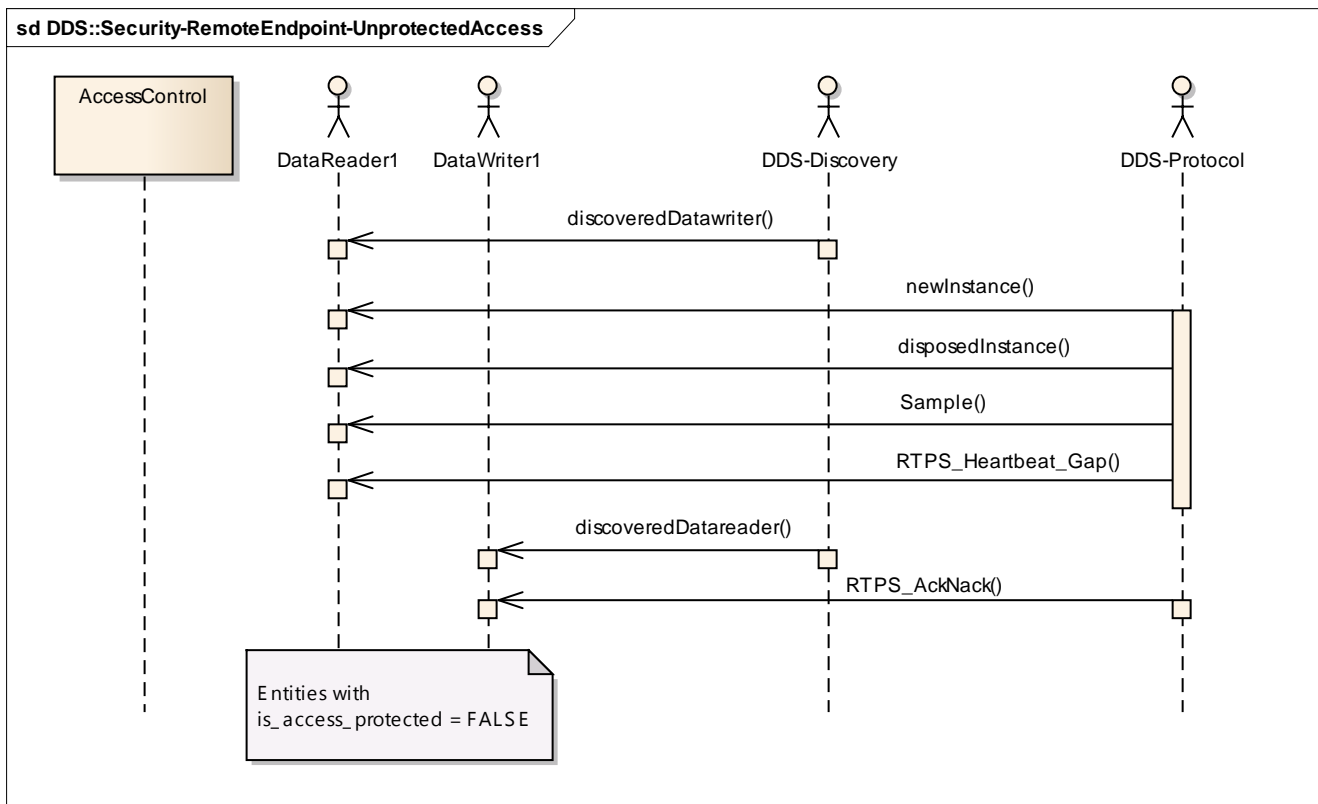
Figure 25 – AccessControl sequence diagram with discovered entities when is_read_protected and is_write_protected are both FALSE

1. DataReader1 discovers via the discovery protocol that a remote `DataWriter` (DataWriter2) on a `Topic` with name **topicName**. The DataReader1 shall not call any operations on the `AccessControl` plugin and shall proceed to match DataWriter2 subject to the matching criteria specified in the DDS and DDS-XTypes specifications.
2. DataReader1 receives a Sample from DataWriter2 with DDS `ViewState` NEW, indicating this is the first sample for that instance received by the `DataReader`. This sample shall be processed according to the DDS specification without any calls to the `AccessControl` plugin.
3. DataReader1 receives a Sample from DataWriter2 with DDS `InstanceState` NOT_ALIVE_DISPOSED, indicating the remote `DataWriter` disposed an instance. This sample shall be processed according to the DDS specification without any calls to the `AccessControl` plugin.
4. DataReader1 receives a Sample from DataWriter2 with DDS `ViewState` NOT_NEW. DataReader1 shall operate according to the DDS and DDS-RTPS specifications without any calls to the `AccessControl` plugin.
5. DataReader1 receives an RTPS `HeartBeat` message or an RTPS `Gap` message from DataWriter2. In both these cases DataReader1 shall operate according to the DDS and DDS-RTPS specifications without any calls to the `AccessControl` plugin.
6. DataWriter1 discovers via the discovery protocol that a remote `DataReader` (DataReader2) on a `Topic` with name **topicName**. DataWriter1 shall not call any operations on the `AccessControl` plugin and shall match DataReader2 subject to the matching criteria specified in the DDS and DDS-XTypes specifications.
7. DataWriter1 receives an RTPS AckNack message from DataReader2. DataWriter1 shall operate according to the DDS and DDS-RTPS specifications without any calls to the `AccessControl` plugin.

The figure below illustrates the behavior relative to discovered endpoints coming from an "Authenticated" `DomainParticipant` that would match local endpoints for which both **is_read_protected** and **is_write_protected** attributes are set to TRUE.
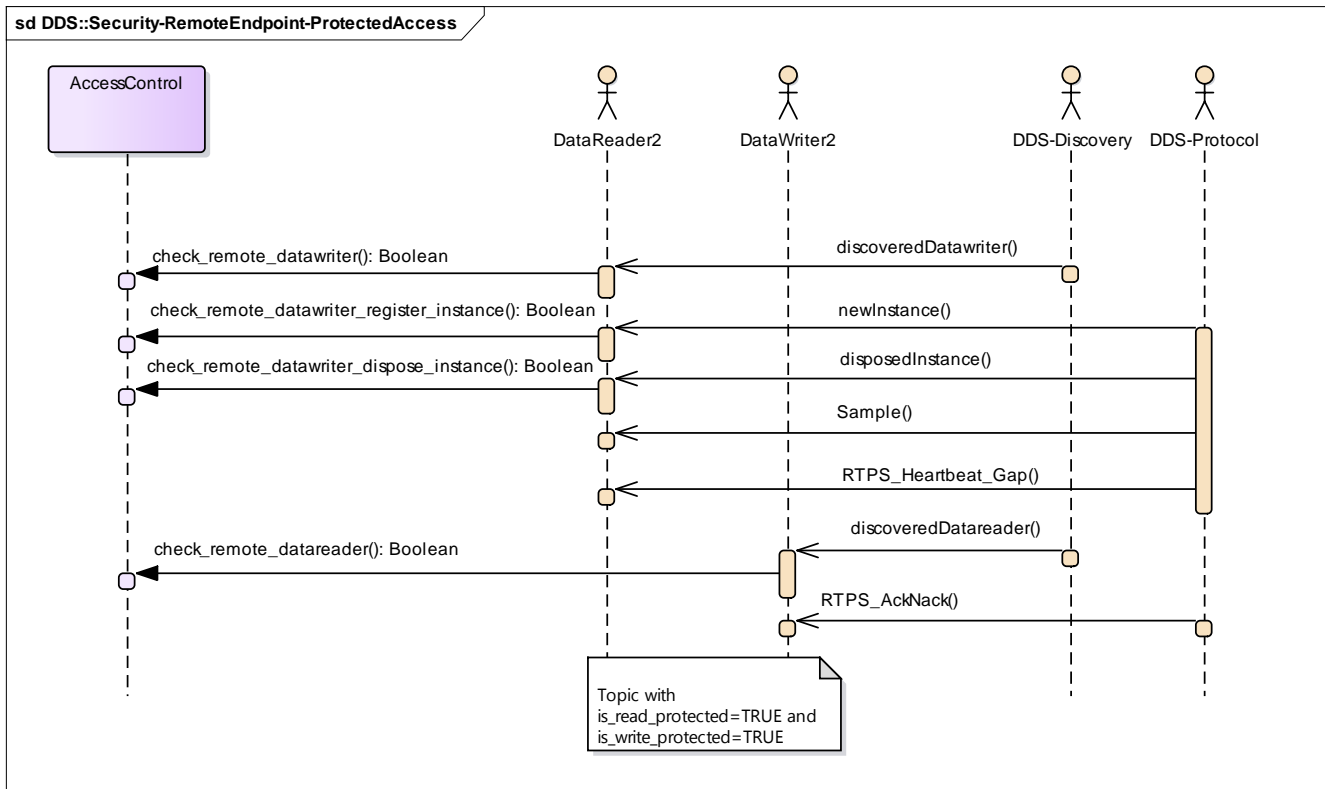
Figure 26 – AccessControl sequence diagram with discovered entities when is_read_protected==TRUE and is_write_protected==TRUE

1. DataReader1 discovers via the discovery protocol a remote `DataWriter` (DataWriter2) on a `Topic` with name ***topicName*** that matches the DataReader1 `Topic` ***topicName***.
2. DataReader1 shall call the operation `check_remote_datawriter` to verify that Participant2 (the `DomainParticipant` to whom DataWriter2 belongs) has the permissions needed to publish the DDS `Topic` with name ***topicName***. As an optional behavior, the same operation can also verify if the DataWriter2 is allowed to tag data with `dataTag` that are associated with it.
   1. If the verification doesn't succeed, the DataWriter2 is ignored.
   2. If the verification succeeds, DataReader1 shall proceed to match DataWriter2 subject to the matching criteria specified in the DDS and DDS-XTypes specifications.
3. DataReader1 receives a Sample from DataWriter2 with DDS ViewState NEW, indicating this is the first sample for that instance received by the `DataReader`. This sample shall be processed according to the DDS specification without any calls to the `AccessControl` plugin.
4. DataReader1 shall call the operation `check_remote_datawriter_register_instance` to verify that Participant2 has the permissions needed to register the instance. If the verification doesn't succeed, the sample shall be ignored.
5. DataReader1 receives a Sample from DataWriter2 with DDS `InstanceState` NOT_ALIVE_DISPOSED, indicating the remote `DataWriter` disposed an instance.
6. DataReader1 shall call the operation `check_remote_datawriter_dispose_instance` to verify that Participant2 has the permissions needed to dispose the instance. If the verification doesn't succeed, the instance disposal shall be ignored.

7. DataReader1 receives a Sample from DataWriter2 with DDS `ViewState` NOT_NEW, indicating this DataReader1 already received samples on that instance. This sample shall be processed according to the DDS specification without any calls to the `AccessControl` plugin.
8. DataReader1 receives an RTPS HeartBeat message or an RTPS `Gap` message from DataWriter2. In both these cases DataReader1 shall operate according to the DDS and DDS-RTPS specifications without any calls to the `AccessControl` plugin.
9. DataWriter1 discovers via the discovery protocol a remote `DataReader` (DataReader2) on a `Topic` with name *topicName* that matches the DataReader1 `Topic` *topicName*.
10. DataWriter1 shall call the operation `check_remote_datareader` to verify that Participant2 (the DomainParticipant to whom DataReader2 belongs) has the permissions needed to subscribe the DDS `Topic` with name *topicName*. As an optional behavior, the same operation can also verify if the DataReader2 is allowed to read data with `dataTag` that are associated with DataWriter1.
   1. If the verification doesn't succeed, DataReader2 is ignored.
   2. If the verification succeeds, DataWriter1 shall proceed to match DataReader2 subject to the matching criteria specified in the DDS and DDS-XTypes specifications.
11. DataWriter1 receives an RTPS `AckNack` message from DataReader2. DataWriter1 shall operate according to the DDS and DDS-RTPS specifications without any calls to the `AccessControl` plugin.

8.8.8 Cryptographic Plugin key generation behavior

Key Generation is potentially needed for:

- The `DomainParticipant` as a whole
- Each `DomainParticipant` match pair
- Each builtin secure endpoint (`DataWriter` or `DataReader`)
- Each builtin secure endpoint match pair
- Each application secure endpoint (`DataWriter` or `DataReader`)
- Each application secure endpoint match pair

8.8.8.1 Key generation for the BuiltinParticipantVolatileMessageSecureWriter and BuiltinParticipantVolatileMessageSecureReader

The *BuiltinParticipantVolatileMessageSecureWriter* and *BuiltinParticipantVolatileMessageSecureReader* endpoints are special in that they are the ones used to securely send the Crypto Tokens. Therefore the key material needed to secure this channel has to be derivable from the `SharedSecret` without having access to Crypto Tokens returned by the `create_local_datawriter_crypto_tokens` or `create_local_datareader_crypto_tokens`. Effectively this means the key material used for key-exchange is always derived from the `SharedSecret`.

For the *BuiltinParticipantVolatileMessageSecureWriter* the creation of the key material necessary to communicate with a matched *BuiltinParticipantVolatileMessageSecureReader* shall complete during the operation `register_matched_remote_datareader` and the DDS middleware shall not call the operation `create_local_datawriter_crypto_tokens` or the operation `set_remote_datareader_crypto_tokens` on the CryptoKeyExchange.

For the ***BuiltinParticipantVolatileMessageSecureReader*** the creation of the key material necessary to communicate with a matched ***BuiltinParticipantVolatileMessageSecureWriter*** shall complete during the operation `register_matched_remote_datawriter` and the DDS middleware shall not call the operation `create_local_datareader_crypto_tokens` or the operation `set_remote_datawriter_crypto_tokens` on the CryptoKeyExchange.

The DDS implementation shall add a property with name "`dds.sec.`builtin_endpoint_name" and value "BuiltinParticipantVolatileMessageSecureWriter" to the `Property_t` passed to the operation `register_local_datawriter` when it registers the ***BuiltinParticipantVolatileMessageSecureWriter*** with the CryptoKeyFactory.

The DDS implementation shall add a property with name "`dds.sec.`builtin_endpoint_name" and value "BuiltinParticipantVolatileMessageSecureReader" to the `Property_t` passed to the operation `register_local_datareader` when it registers the ***BuiltinParticipantVolatileMessageSecureReader*** with the CryptoKeyFactory.

Setting the `Property_t` as described above allows the CryptoKeyFactory to recognize the ***BuiltinParticipantVolatileMessageSecureWriter*** and the ***BuiltinParticipantVolatileMessageSecureReader.***

8.8.8.2 Key generation for the DomainParticipant

For each local `DomainParticipant` that is successfully created the DDS implementation shall call the operation `register_local_participant` on the `KeyFactory`.

For each discovered `DomainParticipant` that has successfully authenticated and has been matched to the local `DomainParticipant` the DDS middleware shall call the operation `register_matched_remote_participant` on the `KeyFactory`. Note that this operation takes as one parameter the SharedSecret obtained from the Authentication plugin.

8.8.8.3 Key generation for the builtin endpoints

For each `DataWriter` belonging to list of "Builtin Secure Endpoints", see 7.4.5, with the exception of the ***BuiltinParticipantVolatileMessageSecureWriter***, the DDS middleware shall call the operation `register_local_datawriter` on the `KeyFactory` to obtain the `DatawriterCryptoHandle` for the builtin `DataWriter`.

For each `DataReader` belonging to list of "Builtin Secure Endpoints", see 7.4.5, with the exception of the ***BuiltinParticipantVolatileMessageSecureReader***, the DDS middleware shall call the operation `register_local_datareader` on the `KeyFactory` to obtain the `DatareaderCryptoHandle` for the corresponding builtin `DataReader`.

For each discovered `DomainParticipant` that has successfully authenticated and has been matched to the local `DomainParticipant` the DDS middleware shall:

1. Call the operation `KeyFactory::register_matched_remote_datawriter` for each local `DataWriter` belonging to the "Builtin Secure Endpoints" passing it the local `DataWriter` and the corresponding remote `DataReader` belonging to the "Builtin Secure Endpoints" of the discovered `DomainParticipant`.
2. Call the operation `KeyFactory::register_matched_remote_datareader` for each local `DataReader` belonging to the "Builtin Secure Endpoints" passing it the local

`DataReader`, the corresponding remote `DataWriter` belonging to the "Builtin Secure Endpoints" of the discovered `DomainParticipant`, and the `SharedSecret` obtained from the Authentication plugin.

8.8.8.4 Key generation for the application-defined endpoints

Recall that for each application-defined (non-builtin) `DataWriter` and `DataReader` successfully created by the DDS Application the DDS middleware has an associated `EndpointSecurityAttributes` object which is the one returned by the `AccessControl::get_datawriter_sec_attributes` or `AccessControl::get_datareader_sec_attributes`.

For each non-builtin `DataWriter` for whom the associated `EndpointSecurityAttributes` object has either the member *is_submessage_protected* or the member *is_payload_protected* set to TRUE, the DDS middleware shall:

1. Call the operation `register_local_datawriter` on the `KeyFactory` to obtain the `DatawriterCryptoHandle` for the `DataWriter`.
2. Call the operation `register_matched_remote_datareader` for each discovered `DataReader` that matches the `DataWriter`.

For each non-builtin `DataReader` for whom the associated `EndpointSecurityAttributes` object has either the member *is_submessage_protected* or the member *is_payload_protected* set to TRUE, the DDS middleware shall:

1. Call the operation `register_local_datareader` on the `KeyFactory` to obtain the `DatareaderCryptoHandle` for the `DataReader`.
2. Call the operation `register_matched_remote_datawriter` for each discovered `DataWriter` that matches the `DataReader`.

8.8.8.5 Limiting message-size overhead caused by receiver specific key material

The use of receiver-specific key material increases the message size in situations where the same encoded message is sent to multiple receivers. For example, when using a multicast transport. In the presence of large numbers or receivers this "per-receiver" overhead may cause a single RTPS submessage with all the receiver-specific authentication codes to exceed the maximum transport MTU. This would cause problems, as RTPS submessages cannot be fragmented.

To overcome this kind of situation implementations may use different strategies.

1. An implementation may limit the number of different receiver-specific key material it generates. For example, it may reuse the same receiver-specific key for multiple receivers. This would limit the overhead at the cost of weakening the origin authentication.
2. An implementation may impose a limit on the number of receiver-specific macs attached to a single message. This would require DDS implementations to construct multiple messages, each with a different set of receiver-specific authentication codes. This use-case is facilitated by the encode_datawriter_submessage and encode_rtps_message CryptoTransform operations.

The selection between and configuration of these choices is implementation specific, as it does not affect interoperability.

8.8.9 Cryptographic Plugin key exchange behavior

Cryptographic key exchange is potentially needed for:

- Each `DomainParticipant` match pair.
- Each builtin secure endpoint match pair.
- Each application secure endpoint match pair.

8.8.9.1 Key Exchange with discovered DomainParticipant

Cryptographic key exchange shall occur between each `DomainParticipant` and each discovered `DomainParticipant` that has successfully authenticated. This key exchange propagates the key material related to encoding/signing/decoding/verifying the whole RTPS message. In other words the key material needed to support the `CryptoTransform` operations `encode_rtps_message` and `decode_rtps_message`.

Given a local DomainParticipant the DDS middleware shall:

1. Call the operation `create_local_participant_crypto_tokens` on the `KeyFactory` for each discovered `DomainParticipant` that has successfully authenticated and has been matched to the local `DomainParticipant`. This operation takes as parameters the local and remote `ParticipantCryptoHandle`.
2. Send the `ParticipantCryptoTokenSeq` returned by operation `create_local_participant_crypto_tokens` to the discovered `DomainParticipant` using ***BuiltinParticipantVolatileMessageSecureWriter***.

The discovered `DomainParticipant` shall call the operation `set_remote_participant_crypto_tokens` passing the `ParticipantCryptoTokenSeq` received by the ***BuiltinParticipantVolatileMessageSecureReader***.

The figure below illustrates the functionality of the Cryptographic KeyExchange plugins with regards to the discovery and match of an authenticated remote `DomainParticipant` entity.
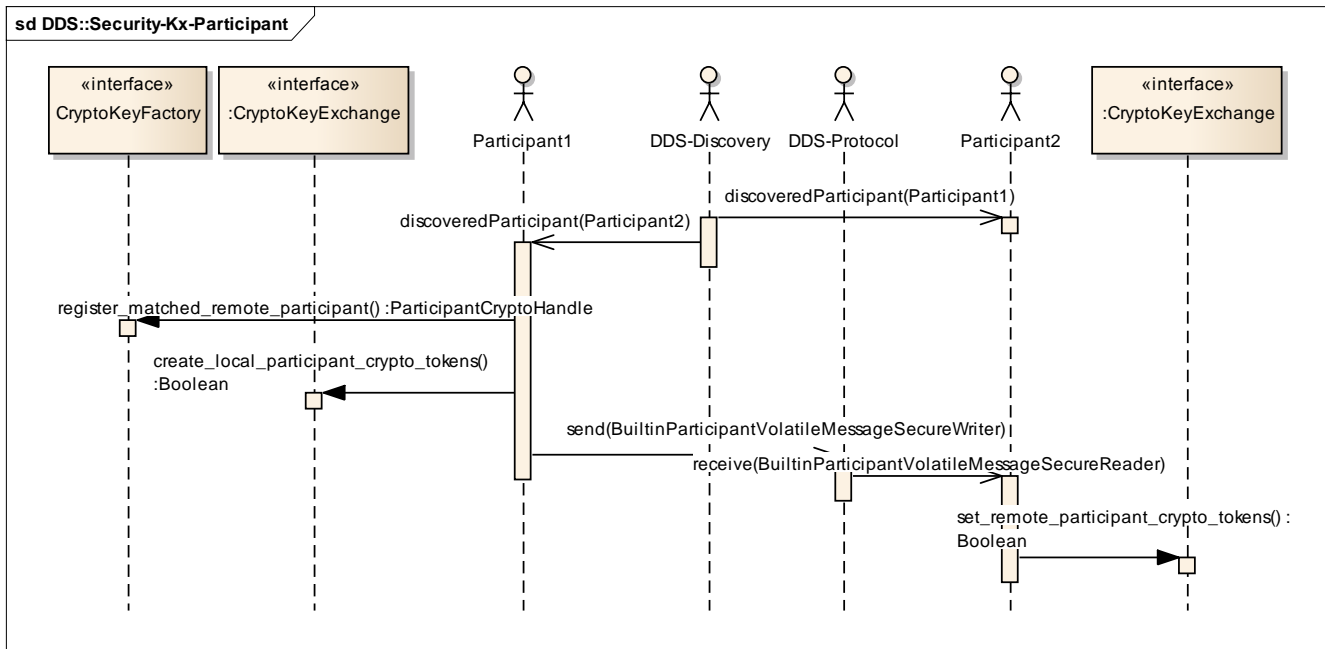
Figure 27 – Cryptographic KeyExchange plugin sequence diagram with discovered DomainParticipant

1. Participant2 discovers the `DomainParticipant` (Participant1) via the DDS discovery protocol. This sequence is not described here as it equivalent to the sequence that Participant1 performs when it discovers Participant2.

2. Participant1 discovers the `DomainParticipant` (Participant2) via the DDS discovery protocol. Participant2 is authenticated and its permissions are checked as described in 8.8.2 and 8.8.6. This is not repeated here. The authentication and permissions checking resulted in the creation of an `IdentityHandle`, a `PermissionsHandle`, and a `SharedSecretHandle` for Participant2.

3. Participant1 calls the operation `register_matched_remote_participant` on the `Cryptographic` plugin (`CryptoKeyFactory` interface) to store the association of the remote identity and the SharedSecret.

4. Participant1 calls the operation `create_local_participant_crypto_tokens` on the `Cryptographic` plugin (`CryptoKeyExchange` interface) to obtain a collection of `CriptoToken` (cryptoTokensParticipant1ForParticipant2) to send to the remote `DomainParticipant` (Participant2).

5. Participant1 sends the collection of `CryptoToken` objects (cryptoTokensParticipant1ForParticipant2) to Participant2 using the ***BuiltinParticipantVolatileMessageSecureWriter***.

6. Participant2 receives the `CryptoToken` objects (cryptoTokensParticipant1ForParticipant2) and calls the operation `set_remote_participant_crypto_tokens()` to register the `CryptoToken` sequence with the `DomainParticipant`. This will enable the `Cryptographic` plugin on Participant2 to decode and verify MACs on the RTPS messages sent by Participant1 to Participant2.

8.8.9.2 Key Exchange with remote DataReader

Cryptographic key exchange shall occur between each builtin secure `DataWriter` and the matched builtin secure `DataReader` entities of authenticated matched `DomainParticipant` entities, see 7.4.5, with the exception of the ***BuiltinParticipantVolatileMessageSecureReader***.

Cryptographic key exchange shall also occur between each application `DataWriter` whose `EndpointSecurityAttributes` object has either the ***is_submessage_protected*** or the ***is_payload_protected*** members set to TRUE, and each of its matched `DataReader` entities.

Given a local `DataWriter` that is either a builtin secure `DataWriter` or an application `DataWriter` meeting the condition stated above the DDS middleware shall:

1. Call the operation `create_local_datawriter_crypto_tokens` on the `KeyFactory` for each matched `DataReader`. This operation takes as parameters the local `DatawriterCryptoHandle` and the remote `DatareaderCryptoHandle`.
2. Send the `DatawriterCryptoTokenSeq` returned by operation `create_local_datawriter_crypto_tokens` to the discovered `DomainParticipant` using ***BuiltinParticipantVolatileMessageSecureWriter***.

The matched `DataReader` shall call the operation `set_remote_datawriter_crypto_tokens` passing the `DatawriterCryptoTokenSeq` received by the ***BuiltinParticipantVolatileMessageSecureReader***.

The figure below illustrates the functionality of the Cryptographic `KeyExchange` plugin with regards to the discovery and match of a local secure `DataWriter` and a matched `DataReader`.
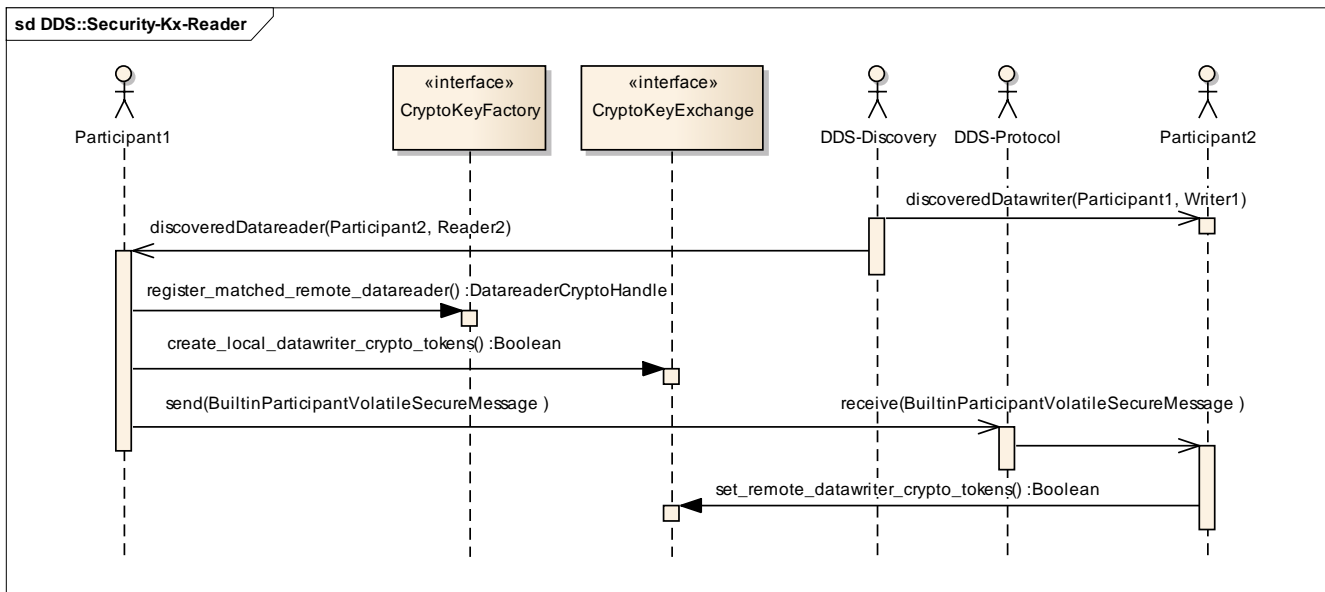
Figure 28 – Cryptographic KeyExchange plugin sequence diagram with discovered DataReader

1. Participant2 discovers a `DataWriter` (Writer1) belonging to Participant1 that matches a local DataReader (Reader2) according to the constraints in the DDS security specification.

2. Participant1 discovers a `DataReader` (Reader2) belonging to Participant2 that matches a local DataWriter (Writer1) according to the constraints in the DDS security specification.

3. Participant1 calls the operation `register_matched_remote_datareader` as stated in 8.8.8.

4. Participant1 calls the operation `create_local_datawriter_crypto_tokens` on the `CryptoKeyExchange` to obtain a collection of `CriptoToken` objects (cryptoTokensWriter1ForReader2).

5. Participant1 sends the collection of `CryptoToken` objects (cryptoTokensWriter1ForReader2) to Participant2 using the ***BuiltinParticipantVolatileMessageSecureWriter***.

6. Participant2 receives the `CryptoToken` objects (cryptoTokensWriter1ForReader2) and calls the operation `set_remote_ datawriter_crypto_tokens()` to register the `CryptoToken` sequence with the `DataWriter (Writer1)`. This will enable the `Cryptographic` plugin on Participant2 to decode and verify MACs on the RTPS submessages and data payloads sent from Writer1to Reader2.

8.8.9.3 Key Exchange with remote DataWriter

Cryptographic key exchange shall occur between each builtin secure `DataReader` and the matched builtin secure `DataWriter` entities of authenticated matched `DomainParticipant` entities, see 7.4.5, with the exception of the ***BuiltinParticipantVolatileMessageSecureReader***.

Cryptographic key exchange shall also occur between each application `DataReader` whose `EndpointSecurityAttributes` object has the ***is_submessage_protected*** member set to TRUE, and each of its matched `DataWriter` entities.

Given a local `DataReader` that is either a builtin secure `DataReader` or an application `DataReader` meeting the condition stated above the DDS middleware shall:

1. Call the operation `create_local_datareader_crypto_tokens` on the `KeyFactory` for each matched `DataWriter`. This operation takes as parameters the local `DatareaderCryptoHandle` and the remote `DatawriterCryptoHandle.`
2. Send the `DatareaderCryptoTokenSeq` returned by operation `create_local_ datareader_crypto_tokens` to the discovered `DomainParticipant` using ***BuiltinParticipantVolatileMessageSecureWriter***.

The matched `DataWriter` shall call the operation `set_remote_datareader_crypto_tokens` passing the `DatareaderCryptoTokenSeq` received by the ***BuiltinParticipantVolatileMessageSecureReader***.

The figure below illustrates the functionality of the Cryptographic `KeyExchange` plugin with regards to the discovery and match of a local secure `DataReader` and a matched `DataWriter`.

Cryptographic key exchange shall occur between each `DataReader` whose `EndpointSecurityAttributes` has the ***is_submessage_protected*** members set to TRUE and each of its matched `DataWriter` entities.
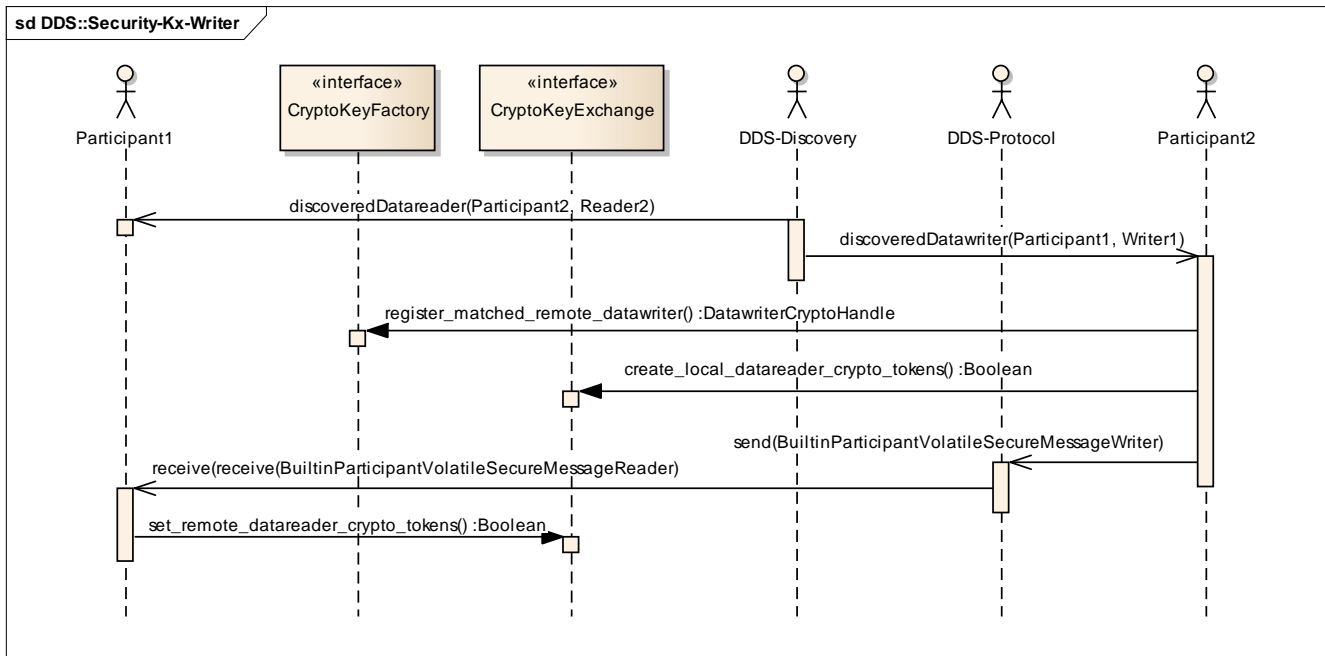
Figure 29 – Cryptographic KeyExchange plugin sequence diagram with discovered DataWriter

1. Participant1 discovers a `DataReader` (Reader2) belonging to Participant2 that matches a local DataWriter (Writer1) according to the constraints in the DDS security specification.

2. Participant2 discovers a `DataWriter` (Writer1) belonging to Participant1 that matches a local DataReader (Reader2) according to the constraints in the DDS security specification.

3. Participant2 calls the operation `register_matched_remote_datawriter` as stated in 8.8.8.

4. Participant2 calls the operation `create_local_datareader_crypto_tokens` on the `CryptoKeyExchange` to obtain a collection of `CriptoToken` objects (cryptoTokensReader2ForWriter1).

5. Participant2 sends the collection of `CryptoToken` objects (cryptoTokensReader2ForWriter1) to Participant1 using the ***BuiltinParticipantVolatileMessageSecureWriter***.

6. Participant1 receives the `CryptoToken` objects (cryptoTokensReader2ForWriter1) and calls the operation `set_remote_ datareader_crypto_tokens()` to register the `CryptoToken` sequence with the `DataWriter` (Writer1). This will enable the `Cryptographic` plugin on Participant1 to decode and verify MACs on the RTPS submessages sent from Reader2 to Writer1.

8.8.10 Cryptographic Plugins encoding/decoding behavior

This sub clause describes the behavior of the DDS implementation related to the `CryptoTransform` interface.

This specification does not mandate a specific DDS implementation in terms of the internal logic or timing when the different operations in the `CryptoTransform` plugin are invoked. The sequence

charts below just express the requirements in terms of the operations that need to be called and their interleaving. This specification only requires that by the time the RTPS message appears on the wire the proper encoding operations have been executed first on each `SerializedPayload` submessage element, then on the enclosing RTPS `Submessage`, and finally on the RTPS `Message`. Similarly by the time a received RTPS `Message` is interpreted the proper decoding operations are executed on the reverse order. First on the encoded RTPS `Message`, then on each set of secured submessages starting with either a `SecureRTPSPrefixSubMsg` or `SecurePrefixSubMsg`, and finally on each `CryptoContent` submessage element.

8.8.10.1 Encoding/decoding of a single writer message on an RTPS message

The figure below illustrates the functionality of the security plugins with regards to encoding the data, Submessages and RTPS messages in the situation where the intended RTPS `Message` contains a single writer RTPS `Submessage`.
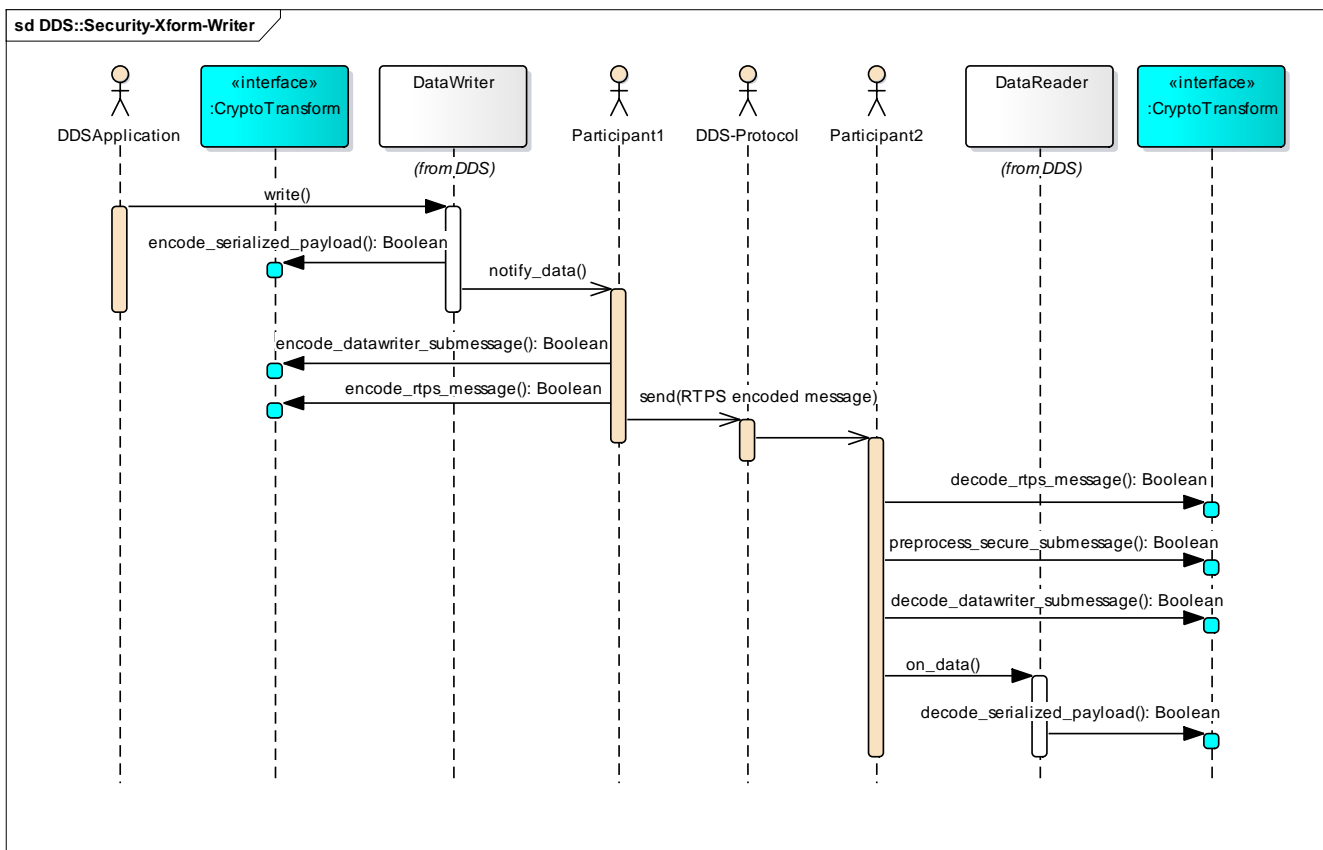
Figure 30 – Cryptographic CryptoTransform plugin sequence diagram for encoding/decoding a single DataWriter submessage

1. The application writes data using a `DataWriter` belonging to Participant1. The DDS implementation serializes the data.

2. The `DataWriter` in Participant1 constructs the `SerializedPayload` RTPS submessage element and calls the operation `encode_serialized_payload`. This operation creates an RTPS `SecData` that protects the `SerializedPayload` potentially encrypting it, adding a MAC and/or digital signature.

3. This step is notional; the specific mechanism depends on the DDS Implementation. Participant1 realizes it is time to send the data written by the DataWriter to a remote `DataReader` in Participant2.

4. Participant1 constructs the RTPS `Data` Submessage to send to the DataReader and calls the operation `encode_datawriter_submessage` to transform the original `Data` submessage to a set of secure submessages (`SecurePrefixSubMsg`, original plain text submessage or `SecureBodySubMsg`, and optional `SecurePostfixSubMsg`). This same transformation would be applied to any `DataWriter` submessage (`Data`, `Gap`, `Heartbeat`, `DataFrag`, `HeartbeatFrag`). The `encode_datawriter_submessage` receives as parameters the `DatawriterCryptoHandle` of the `DataWriter` and a list of `DatareaderCryptoHandle` for all the `DataReader` entities to which the message will be sent. Using a list allows the same set of secure submessages to be sent to all those `DataReader` entities.

5. Participant1 constructs the RTPS `Message` it intends to send to the DataReader (or readers). It then calls `encode_rtps_message` to transform the original RTPS Message into a new "encoded" RTPS Message with the same RTPS header and a set of secure submessages protecting the contents of the original RTPS Message. The `encode_rtps_message` receives as parameters the `ParticipantCryptoHandle` of the sending `DomainParticipant` (Participant1) and a list of `ParticipantCryptoHandle` for all the `DomainParticipant` entities to which the message will be sent (Participant2). Using a list enables the `DomainParticipant` to send the same message (potentially over multicast) to all those `DomainParticipant` entities.

6. Participant1 sends the new "encoded" RTPS Message obtained as a result of the previous step to Participant2.

7. Participant2 receives the "encoded" RTPS Message. Participant2 parses the message and detects a `SecureRTPSPrefixSubMsg`. This indicates it shall call the operation `decode_rtps_message` to to process the prefix, body and optional postfix submessage. If `decode_rtps_message` is successful, the result is an RTPS Message that can be processed further.

8. Participant2 parses the RTPS Message resulting from the previous step and encounters an RTPS `SecurePrefixSubMsg`. This indicates it shall call the operation `prepare_rtps_submessage` to determine whether this is a Writer submessage or a Reader submessage and obtain the `DatawriterCryptoHandle` and `DatareaderCryptoHandle` handles it needs to decode the message. This function determines it is a Writer submessage.

9. Participant2 calls the operation `decode_datawriter_submessage` passing in a data stream that includes the `SecurePrefixSubMsg`, a plain text submessage or a `SecureBodySubMsg`, and an optional `SecurePostfixSubMsg`. The `decode_datawriter_submessage` operation also requires the `DatawriterCryptoHandle` and `DatareaderCryptoHandle` obtained in the previous step. The operation, if successful, will return the original `Data` submessage that

was input to `encode_datawriter_submessage` on the DataWriter side. From the `Data` submessage the DDS implementation extracts the `CryptoContent` submessage element.

10. This step is notional; the specific mechanism depends on the DDS Implementation. Participant2 realizes it is time to notify the DataReader and retrieve the actual data sent by the DataWriter.

11. Participant2 calls `decode_serialized_payload` passing in the RTPS `CryptoContent` and obtains the original `SerializedPayload` submessage element was the input to the `encode_serialized_payload` on the DataWriter side. This operation takes as arguments the `DatawriterCryptoHandle` and `DatareaderCryptoHandle` obtained in step 8.

8.8.10.2 Encoding/decoding of multiple writer messages on an RTPS message

The figure below illustrates the functionality of the security plugins in the situation where the intended RTPS message contains a multiple `DataWriter` RTPS `Submessages`, which can represent multiple samples, from the same `DataWriter` or from multiple DataWriter entities, as well as, a mix of `Data`, `Heartbeat`, `Gap`, and any other `DataWriter` RTPS `Submessage` as defined in 7.3.1.
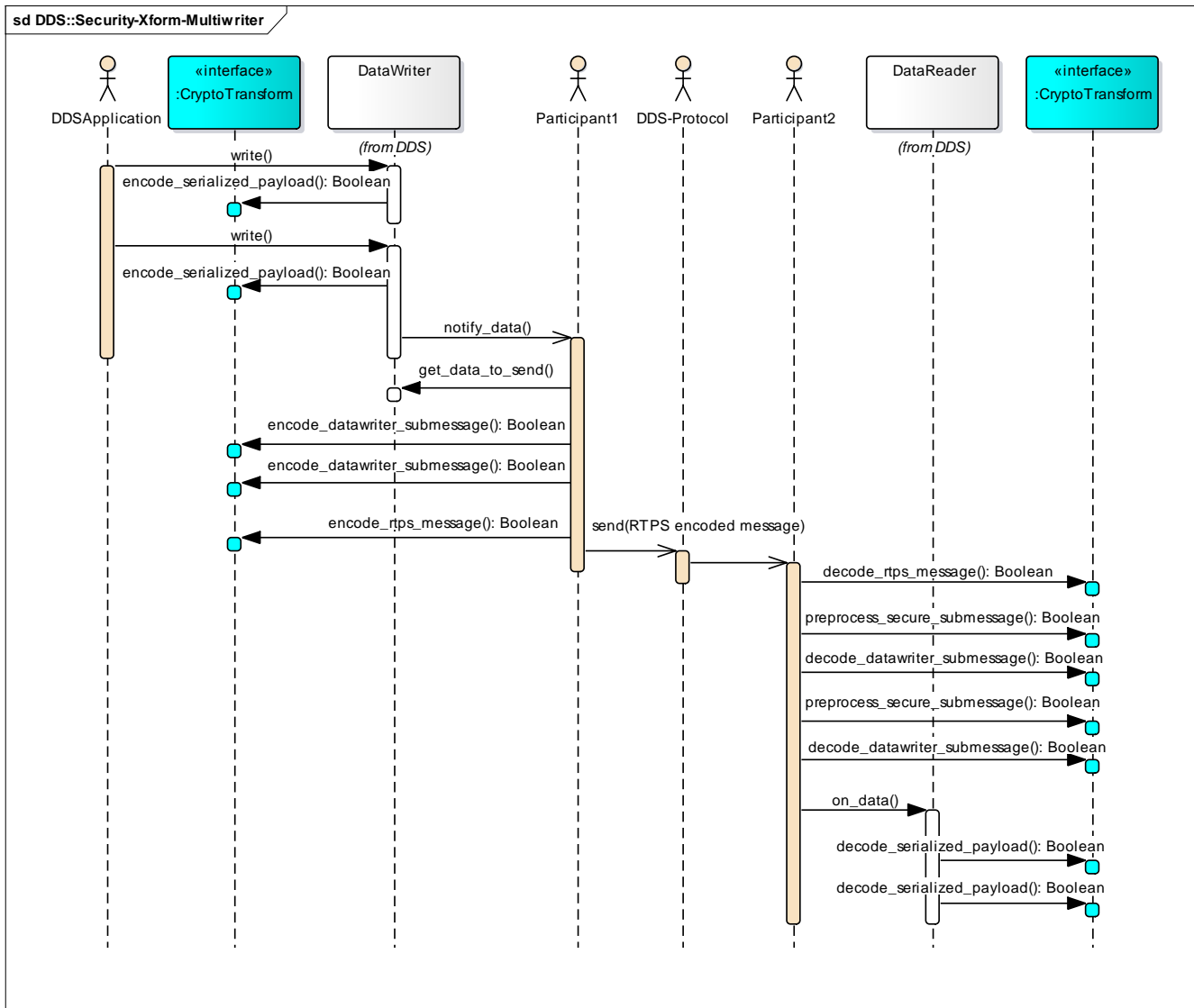
Figure 31 – Cryptographic CryptoTransform plugin sequence diagram for encoding/decoding multiple DataWriter submessages

The steps followed to encode and decode multiple DataWriter Submessages within the same RTPS message are very similar to the ones used for a single Writer message. The only difference is that the writer side can create multiple RTPS Submessages. In this case, Participant1 creates two Data Submessages and a Heartbeat Submessage, transforms each separately using the `encode_datawriter_submessage`, places them in the same RTPS message and then transforms the RTPS Message containing all the resulting secured submessages using `encode_rtps_message`.

The steps followed to decode the message are the reverse ones.

Note that the DataWriter entities that are sending the submessages and/or the DataReader entities that are the destination of the different Submessages may be different. In this situation each call to `encode_serialized_payload()`, `encode_datawriter_submessage()`, `decode_datawriter_submessage()`, and `encode_serialized_payload()`, shall receive the proper `DatawriterCryptoHandle` and `DatareaderCryptoHandle` handles.

8.8.10.3 Encoding/decoding of multiple reader messages on an RTPS message

The figure below illustrates the functionality of the security plugins in the situation where the intended RTPS message contains multiple `DataReader` RTPS submessages from the same `DataReader` or from multiple `DataReader` entities. These include `AckNack` and `NackFrag` RTPS `Submessages` as defined in 7.3.1.
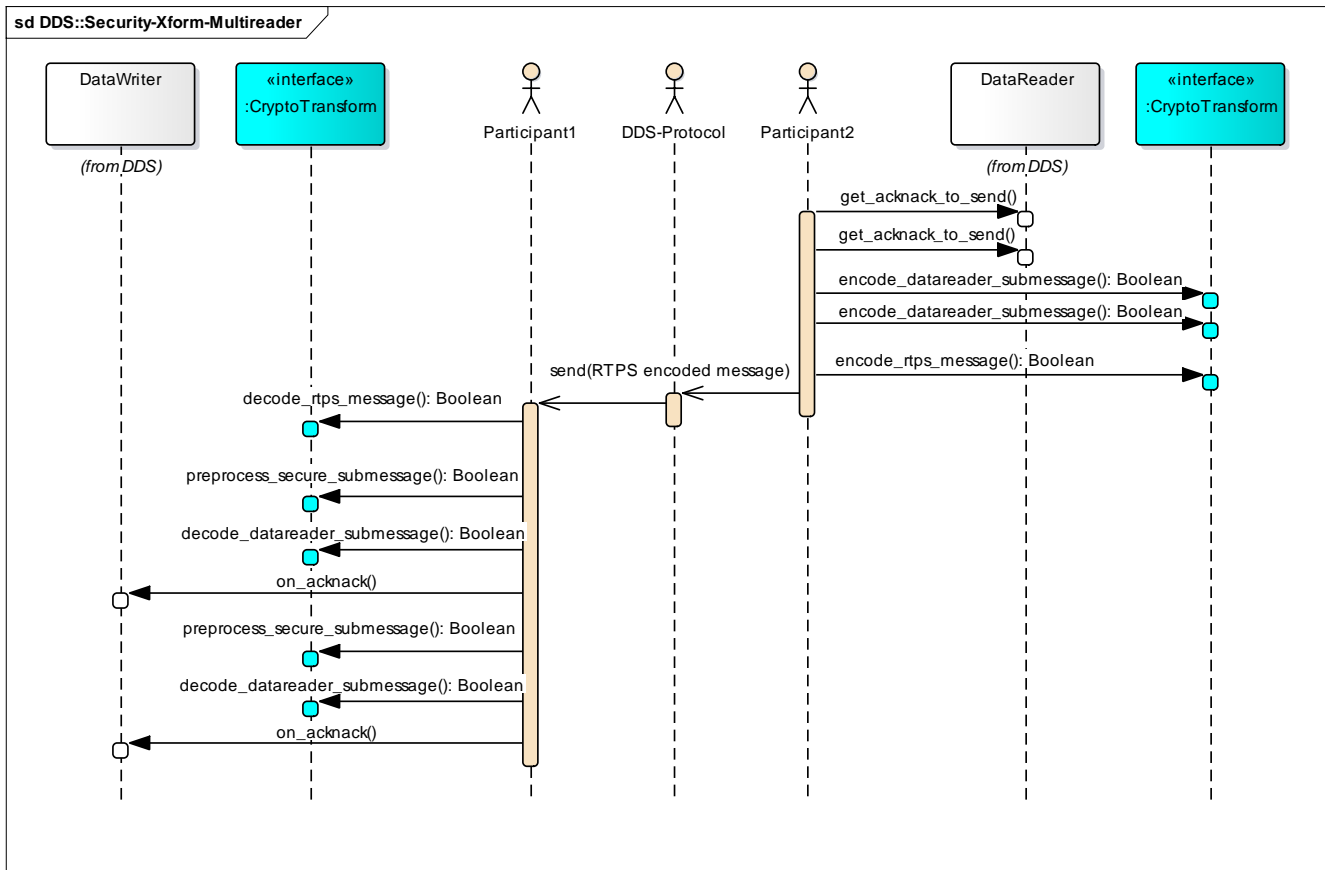
Figure 32 -- Cryptographic CryptoTransform plugin sequence diagram for encoding/decoding multiple DataReader submessages

1. This step is notional; the specific mechanism depends on the DDS Implementation. Participant2 realizes it is time to send an `AckNack` or `NackFrag` submessage from `DataReader` to a remote `DataWriter`.

2. Participant2 constructs the `AckNack` (or any other `DataReader` RTPS `Submessage`) and calls the operation `encode_datareader_submessage`. This operation creates creates multiple submessages: a `SecurePrefixSubMsg`, a plain text submessage or a `SecureBodySubMsg`, and optionally a `SecurePostfixSubMsg`. This operation shall receive as parameter the `DatareaderCryptoHandle` of the `DataReader` that sends the submessage and a list of `DatawriterCryptoHandle` handles of all the `DataWriter` entities to which the `Submessage` will be sent.

3. Step 2 may be repeated multiple times constructing various secured submessages from different `DataReader` RTPS `Submessages`. Different submessages may originate on different `DataReader` entities and/or be destined for different `DataWriter` entities. On each case the `encode_datareader_submessage` operation shall receive the

DatareaderCryptoHandle and list of DatawriterCryptoHandle that correspond to the source and destinations of that particular Submessage.

4. Participant2 constructs the RTPS Message that contains the submessages obtained as a result of the previous steps. It shall then call encode_rtps_message to transform the "original" RTPS Message into a SecureRTPSPrefixSubMsg followed by either 1) an INFO_SRC SubMsg and the contents of the RTPS Message or 2) a SecureBodySubMsg (with INFO_SRC and encoded content), and finally a SecureRTPSPostfixSubMsg.

5. Participant2 sends the "encoded" RTPS Message to Participant1 (and any other destination DomainParticipant).

6. Participant1 receives the "encoded" RTPS Message. Participant parses the message and detects an RTPS SecureRTPSPrefixSubMsg. This indicates it should call the operation decode_rtps_message to process the prefix, body and optional postfix submessage. If decode_rtps_message is successful, the result is an RTPS Message that can be processed further.

7. Participant1 parses the RTPS Message resulting from the previous step and encounters an RTPS SecurePrefixSubMsg. This indicates it shall call the operation preprocess_secure_submessage to determine whether this is a Writer submessage or a Reader submessage and obtain the DatawriterCryptoHandle and DatareaderCryptoHandle handles it needs to decode the message. This function determines it is a DataReader submessage.

8. Participant1 calls decode_datareader_submessage passing in a data stream that includes the SecurePrefixSubMsg, a plain text submessage or a SecureBodySubMsg, and an optional SecurePostfixSubMsg. The decode_datareader_submessage operation also requires the DatawriterCryptoHandle and DatareaderCryptoHandle obtained in the previous step. The operation, if successful, will return the original AckNack (or proper DataReader submessage) submessage that was input to encode_datareader_submessage on the DataReader side.

9. This step is notional; the specific mechanism depends on the DDS Implementation. Participant1 realizes it is time to notify the DataReader of the Acknowledgment, negative acknowledgment or whatever the DataReader Submessage indicated.

10. Each RTPS SecurePrefixSubMsg encountered within the RTPS Message is processed in this same way. The operation preprocess_rtps_submessage is first invoked and if it indicates it is a DataReader submessage Participant1 shall call decode_datareader_submessage() on the submessage.

8.8.10.4 Encoding/decoding of reader and writer messages on an RTPS message

The figure below illustrates the functionality of the security plugins with regards to encoding the data, Submessages and RTPS messages in the situation where the intended RTPS message contains multiple RTPS Submessages which can represent a mix of different kinds of DataWriter and DataReader submessages such as Data, Heartbeat, Gap, AckNack, NackFrag and any other RTPS Submessage as defined in 7.3.1.
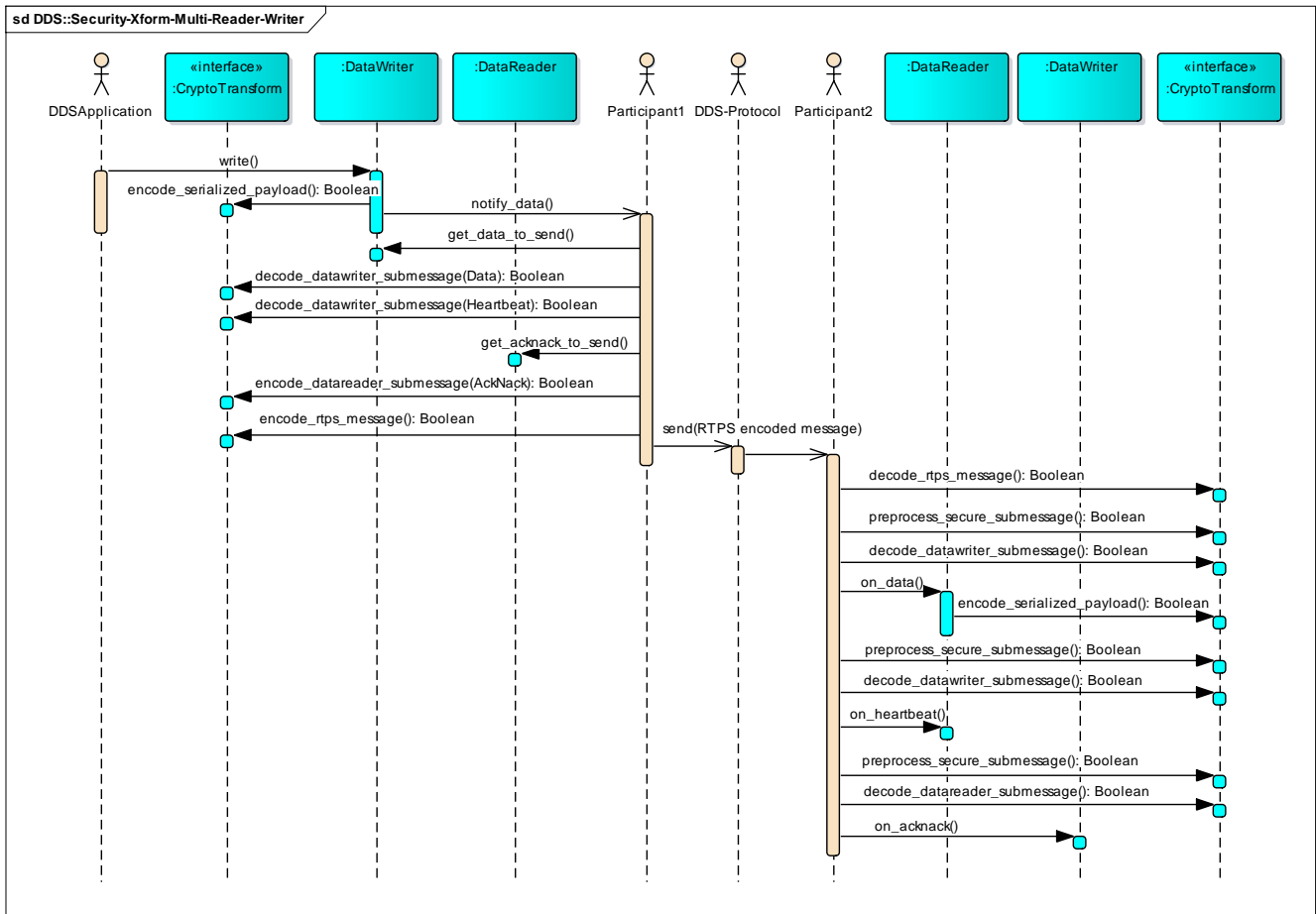
Figure 33 – Cryptographic CryptoTransform plugin sequence diagram for encoding/decoding multiple DataWriter and DataReader submessages

1. The application writes data using a `DataWriter` belonging to Participant1. The DDS implementation serializes the data.

2. The `DataWriter` in Participant1 constructs the `SerializedPayload` RTPS submessage element and calls the operation `encode_serialized_payload`. This operation creates an RTPS `SecData` that protects the `SerializedPayload` potentially encrypting it, adding a MAC and/or digital signature.

3. This step is notional; the specific mechanism depends on the DDS Implementation. Participant1 realizes it is time to send the data written by the `DataWriter` to a remote `DataReader`.

4. Participant1 constructs the RTPS `Data` Submessage that it will send to the `DataReader` and calls the operation `encode_datawriter_submessage` to transform the original `Data` submessage to a set of secured submessages.

5. This step is notional. The specifics will depend on the DDS Implementation. Participant1 decides it needs to send a `Heartbeat` submessage along with the `Data` submessage. It constructs the RTPS `Heartbeat` submessage and calls the operation `encode_datawriter_submessage()` to transform the original `Heartbeat` submessage to a set of secured submessages.

6. This step is notional. The specific mechanism depends on the DDS Implementation. Participant1 decides it also wants to include an RTPS `AckNack` submessage from a DataReader that also belongs to Participant1 into the same RTPS Message because it is destined to the same Participant2.

7. Participant1 constructs the RTPS `AckNack` submessage and calls `encode_datareader_submessage` to transform the original `AckNack` submessage to a set of secured submessages.

8. Participant1 constructs the RTPS Message that contains the submessages obtained as a result of the previous steps. It shall then call `encode_rtps_message`. To transform the "original" RTPS Message into `SecureRTPSPrefixSubMsg` followed by either 1) an `INFO_SRC` SubMsg and the contents of the RTPS `Message` or 2) a `SecureBodySubMsg` (with `INFO_SRC` and encoded content), and finally a `SecureRTPSPostfixSubMsg`.

9. Participant1 sends the "encoded" RTPS `Message` to Participant2 (and any other destination DomainParticipant).

10. Participant2 receives the "encoded" RTPS Message. Participant2 parses the message and detects an RTPS `SecureRTPSPrefixSubMsg`. This indicates it should call the operation `decode_rtps_message` to to process the prefix, body and optional postfix submessage. If `decode_rtps_message` is successful, the result is an RTPS Message that can be processed further.

11. Participant2 parses the RTPS Message resulting from the previous step and encounters an RTPS `SecurePrefixSubMsg`. This indicates it shall call `preprocess_secure_submessage` to determine whether this is a `Writer` submessage or a `Reader` submessage and obtain the `DatawriterCryptoHandle` and `DatareaderCryptoHandle` handles it needs to decode the message. This function determines it is a `DataWriter` submessage.

12. Participant1 calls the operation `decode_datawriter_submessage`, passing in a data stream that includes the `SecurePrefixSubMsg`, a plain text submessage or a `SecureBodySubMsg`, and an optional `SecurePostfixSubMsg`. The `decode_datawriter_submessage` operation also requires the `DatawriterCryptoHandle` and `DatareaderCryptoHandle` obtained in the previous step. The operation, if successful, will return the original DataWriter submessage that was input to `encode_datawriter_submessage` on the Participant1 side.

13. This step is notional; the specific mechanism depends on the DDS Implementation. The Participant2 realizes it is time to notify the `DataReader` of the arrival of data.

14. Participant2 calls `decode_serialized_payload` passing in the RTPS `CryptoContent` and obtains the original `SerializedPayload` submessage element was the input to the `encode_serialized_payload` on the Participant1 side. This operation takes as arguments the `DatawriterCryptoHandle` and `DatareaderCryptoHandle` obtained in the step 11.

15. Step 11 is repeated. It is again determined that the next set of secured submessages are a `DataWriter` submessage and the proper `DatawriterCryptoHandle` and `DatareaderCryptoHandle` handles are retrieved.

16. Step 12 is repeated. Participant2 calls `decode_datawriter_submessage` passing in a data stream that includes the `SecurePrefixSubMsg`, a plain text submessage or a `SecureBodySubMsg`, and an optional `SecurePostfixSubMsg`. This transforms the submessages into the original `Heartbeat` submessage.

17. This step is notional; the specific mechanism depends on the DDS Implementation. Participant2 notifies DataReader of the `Heartbeat`.

18. Step 11 is repeated. It is determined that the next set of submessages are a DataReader submessage and the proper `DatawriterCryptoHandle` and `DatareaderCryptoHandle` handles are retrieved.

19. Participant2 calls `decode_datareader_submessage` passing in a data stream that includes the `SecurePrefixSubMsg`, a plain text submessage or a `SecureBodySubMsg`, and an optional `SecurePostfixSubMsg`. The result of this operation is the original `AckNack` submessage that was the input to the `encode_datareader_submessage` on Participant1. This operation takes as arguments the `DatawriterCryptoHandle` and `DatareaderCryptoHandle` obtained in the previous step.

20. This step is notional; the specific mechanism depends on the DDS Implementation. Participant2 notifies `DataWriter` of the `AckNack`.

9 Builtin Plugins

9.1 Introduction

This specification defines the behavior and implementation of at least one builtin plugin for each kind of plugin. The builtin plugins provide out-of-the-box interoperability between implementations of this specification.

The builtin plugins are summarized in the table below:

Table 43 – Summary of the Builtin Plugins

SPI	*Plugin Name*	*Description*
Authentication	DDS:Auth:PKI-DH	Uses PKI with a pre-configured shared Certificate Authority. RSA or DSA and Diffie-Hellman for authentication and key exchange.
AccessControl	DDS:Access:Permissions	Permissions document signed by shared Certificate Authority
Cryptography	DDS:Crypto:AES-GCM-GMAC	AES-GCM (AES using Galois Counter Mode) for encryption. AES-GMAC for message authentication.
DataTagging	DDS:Tagging:DDS_Discovery	Send Tags via Endpoint Discovery
Logging	DDS:Logging:DDS_LogTopic	Logs security events to a dedicated DDS Log Topic

9.2 Requirements and Priorities (Non-Normative)

The selection of the builtin plugins was driven by several functional, as well as, non-functional requirements, as described below.

Most DDS users surveyed consider the following functional requirements as essential elements of a secure DDS middleware:

- Authentication of applications (DDS Domain Participants) joining a DDS Domain.

- Access control of applications subscribing to specific data at the Domain and Topic level.

- Message integrity and data-origin authentication.

- Encryption of a data sample using different encryption keys for different Topics.

In addition to these essential needs, many users also required that secure DDS middleware should provide for:

- Sending digitally signed data samples.

- Sending data securely over multicast.

- Tagging data.

- Integrating with open standard security plugins.

Other functional requirements which are considered useful but less common were:

- Access control to certain samples within a Topic but not others, with access rights being granted according to the data-sample contents or the data-sample key.

- Access control to certain attributes within a data sample but not others, such that certain DataReader entities can only observe a subset of the attributes as defined by their permissions.

- Permissions that control which QoS might be used by a specific DDS Entity: DomainParticipant, Publisher, DataWriter, Subscriber, or DataReader.

The primary non-functional requirements that informed the selection of the builtin plugins are:

- Performance and Scalability.

- Robustness and Availability.

- Fit to the DDS Data-Centric Information Model.

- Leverage and reuse of existing security infrastructure and technologies.

- Ease of use while supporting common application requirements.

9.2.1 Performance and Scalability

DDS is commonly deployed in systems that demand high performance and need to scale to large numbers of processes and computers. Different applications vary greatly in the number of processes, Topics, and/or data-objects belonging to each Topic.

The policy enforcement/decision points as well as the transformations (cipher, decipher, hash) performed by the plugins should not adversely degrade system performance and scalability beyond what is tolerable and strictly needed. In practice this means several things for the builtin plugins:

- The use of Asymmetric Key Cryptography shall be limited to the discovery, authentication, session and shared-secret establishment phase (i.e., when a Participant discovers another Participant, a DataReader and matching DataWriter). To the extent possible it shall not be used in the critical path of data distribution.

- The use of ciphers, HMACs, or digital signatures shall be selectable on a per stream (Topic) basis. In case of encryption, symmetric ciphers should be used for the application data.

- It shall be possible to provide integrity via HMAC techniques without also requiring the data to be ciphered.

- Multicast shall be supported even for ciphered data.

9.2.2 Robustness and Availability

DDS is deployed in mission-critical systems, which must continue to operate 24/7 despite partial system malfunction. DDS also operates in fielded environments where specific components or systems may be subject to accidental failure or active attack. DDS provides a highly robust infrastructure due to the way the communication model and protocols are defined as they can be (and commonly are)

implemented in a peer-to-peer fashion without any centralized services. For this reason, many DDS implementations have no single points of failure.

The builtin plugins should not negate these desirable properties present in the underlying DDS middleware infrastructure.

In practice, this means that:

- Centralized policy decision points or services should be avoided.

- The individual DDS DomainParticipant components should be self-contained and have what they need to operate securely even in the presence of system partitions.

- Multi-party key agreement protocols shall be avoided because they can be easily disrupted by disrupting just one party.

- Security tokens and keys should be compartmentalized as much as possible such that compromise of an application component is contained to that component itself. For example, selection of a system-wide secret key for the whole Domain or even for a Topic should be avoided.

9.2.3 Fitness to the DDS Data-Centric Model

Application developers that use DDS think in terms of the data-centric elements that DDS provides. That is, they think first and foremost about the Domains (global data spaces) the application must join and the Topics that the application needs to read and write. Therefore, the builtin plugins should offer the possibility to control access with this level of granularity.

Users of DDS also think about the data objects (keyed instances) they read and write, the ability to dispose instances, filter by content, set QoS, and so forth. While it may be useful to offer ways to provide access controls to this as well, it was considered of lesser priority and potentially conflicting with the goal of ease of configurability and maintainability.

The semantics of DDS communications require that individual samples can be consumed independently of each other. Depending on the QoS policy settings samples written by a single DataWriter may be received and processed out of order relative to the order sent, or may be received with intermediate gaps resulting from best-effort communication (if selected), or may be filtered by content, time, or history, etc. For this reason, any encryption and/or digital signature applied to a sample should be able to be processed in isolation, without requiring the receiver to maintain a specific context reconstructed from previous samples.

9.2.4 Leverage and Reuse of Existing Security Infrastructure and Technologies

To the extent possible, it is desirable that the builtin plugins leverage and reuse existing IA technology and tools. This not only reduces the barrier of entry for implementers of the specification, but also more importantly enhances the quality of the result by allowing the use of proven, peer-reviewed, and/or already certified approaches. The builtin plugins leverage existing standards and tools for PKI, ciphers, hashing and digital signing. To the extent possible, ideas and approaches from existing protocols for key management and secure multicast are also leveraged, although where appropriate they have been adapted to the data-centric communications model of DDS and the DDS-RTPS wire protocol.

9.2.5 Ease-of-Use while Supporting Common Application Requirements

It is anticipated that specialized applications may need to develop their own security plugins to either integrate existing security infrastructure or meet specialized requirements. Therefore the primary consumers of the builtin plugins will be users who want to secure their systems but not have complex needs or significant legacy components. Under these conditions, ease-of-use is essential. A security infrastructure that is too hard to configure or too complex to understand or maintain is less likely to be used, or may be used wrongly, resulting in systems that are less secure overall.

The builtin plugins balance rich functionality and ease-of-use, providing for the most common use cases, in a manner that is easy to understand and use correctly.

9.3   Builtin Authentication: DDS:Auth:PKI-DH

This builtin authentication plugin is referred to as the "DDS:Auth:PKI-DH".

The DDS:Auth:PKI-DH plugin implements authentication using a trusted `Certificate Authority` (CA). It performs mutual authentication between discovered participants using the RSA or ECDSA Digital Signature Algorithms [11] and establishes a shared secret using Diffie-Hellman (DH) or Elliptic Curve Diffie-Hellman (ECDH) Key Agreement Methods [12].

The CA could be an existing one. Or a new one could be created for the purpose of deploying applications on a DDS `Domain`. The nature or manner in which the CA is selected is not important because the way it is used enforces a shared recognition by all participating applications.

Prior to a `DomainParticipant` being enabled the DDS:Auth:PKI-DH plugin associated with the `DomainParticipant` must be configured with three things:

1. The `X.509 Certificate` that defines the `Shared Identity CA`. This certificate contains the `Public Key` of the CA.

2. The `Private Key` of the `DomainParticipant`.

3. An `X.509 Certificate` that chains up to the `Shared Identity CA`, that binds the `Public Key` of the `DomainParticipant` to the `Distinguished Name` (subject name) for the `DomainParticipant`.

9.3.1 Configuration

The builtin authentication plugin shall be configured using the `PropertyQosPolicy` of the `DomainParticipantQos`. The specific properties used are described in Table 44 below.

Table 44 – Properties used to configure the builtin Authentication plugin

Property Name (all properties have "dds.sec.auth" prefix)	Property Value (all these properties shall have propagate set to FALSE) URI syntax follows IETF RFC 3986. URI "data" schema follows IETF RFC 2397 URI "pkcs11" schema follows IETF RFC 7512 Vendors may support additional schemas
identity_ca	URI to the X509 certificate [39] of the Identity CA. Supported URI schemes: file, data, pkcs11 The **file** and **data** schemas shall refer to a X.509 v3 certificate (see X.509 v3 ITU-T Recommendation X.509 (2005) [39]) in PEM format. Examples: file:identity_ca.pem file:/home/myuser/identity_ca.pem data:,-----BEGIN CERTIFICATE----- MIIC3DCCAcQCCQCWE5x+Z … PhovK0mp2ohhRLYI0ZiyYQ== -----END CERTIFICATE----- pkcs11:object=MyIdentityCACert;type=cert
private_key	URI to access the private Private Key for the DomainParticipant Supported URI schemes: file, data, pkcs11 pkcs11 URI follows IETF RFC 7512 "The PKCS #11 URI Scheme" Examples: file:identity_ca_private_key.pem file:/home/myuser/identity_ca_private_key.pem file:identity_ca_private_key.pem?password=OpenSesame data:,-----BEGIN RSA PRIVATE KEY----- MIIEpAIBAAKCAQEA3HIh...AOBaaqSV37XBUJg== -----END RSA PRIVATE KEY----- pkcs11:object=MyParticipantPrivateKey;type=private?pin-value=OpenSesame
password	A password used to decrypt the private_key. The value of the password property shall be interpreted as the Base64 encoding of the AES-128 key that shall be used to decrypt the private_key using AES128-CBC. If the password property is not present, then the value supplied in the private_key property must contain the unencrypted private key. The password property is only used if the private_key is provided with a "file:" or a "data:" URI. It does not apply to private keys supplied with the "pkcs11:" URI.

identity_certificate	URI to a X509 certificate signed by the IdentityCA in PEM format containing the signed public key for the DomainParticipant Supported URI schemes: file, data, pkcs11 Examples: file:participant1_identity_cert.pem data:,-----BEGIN CERTIFICATE----- MIIDjjCCAnYCCQDCEu9...6rmT87dhTo= -----END CERTIFICATE----- pkcs11:object=MyParticipantIdentityCert;type=cert

9.3.1.1 Identity CA Certificate

The certificate used to configure the public key of the Identity CA.

The certificate shall be the X.509 v3 Certificate [39] of the issuer of the Identity Certificates in section 9.3.1.3. The certificate can be self-signed if it is a root CA or signed by some other CA public key if it is a subordinate CA. Regardless of this the Public Key in the Certificate shall be accepted as the one for the Identity CA trusted to sign `DomainParticipant` Identity Certificates, see 9.3.1.3.

The public key of the CA shall be either a 2048-bit RSA key [44] or else a 256-bit Elliptic Curve Key for the prime256v1 curve [41], also known as the NIST P-256 curve [42].

The Identity CA Certificate shall be provided to the plugins using the `PropertyQosPolicy` on the `DomainParticipantQos` as specified in Table 44.

9.3.1.2 Private Key

The Private Key associated with the `DomainParticipant`. It may be either a 2048-bit RSA private key or a 256-bit Elliptic Curve Key for use with the prime256v1 curve [41].

The Private Key shall be provided to the plugins using the `PropertyQosPolicy` on the `DomainParticipantQos` as specified in Table 44.

9.3.1.3 Identity Certificate

An X.509 v3 Certificate [39] that chains up to the Identity CA (see 9.3.1.1). The Identity Certificate binds the Public Key of the `DomainParticipant` to the Distinguished Name (subject name) for the `DomainParticipant`.

9.3.2 DDS:Auth:PKI-DH Types

This sub clause specifies the content and format of the `Credential` and `Token` objects used by the DDS:Auth:PKI-DH plugin.

`Credential` and `Token` attributes left unspecified in this specification shall be understood to not have any required values in this specification. These attributes shall be handled according to the following rules:

- Plugin implementations may place data in these attributes as long as they also include a property attribute that allows the implementation to unambiguously detect the presence and interpret these attributes.
- Attributes that are not understood shall be ignored.
- `Property_t` and `BinaryProperty_t` names shall comply with the rules defined in 7.2.1 and 7.2.2, respectively.

The content of the `Handle` objects is not specified as it represents references to internal state that is only understood by the plugin itself. The DDS Implementation only needs to hold a reference to the returned `Handle` objects returned by the plugin operations and pass these `Handle` references to other operations.

9.3.2.1 DDS:Auth:PKI-DH IdentityToken

The DDS:Auth:PKI-DH plugin shall set the attributes of the `IdentityToken` object as specified in the table below:

Table 45 – IdentityToken class for the builtin Authentication plugin

Attribute name	Attribute value	
class_id	"DDS:Auth:PKI-DH:1.0"	
properties (The presence of each of properties is optional)	*name*	*value*
	dds.cert.sn	The subject name of the Identity Certificate.
	dds.cert.algo	"RSA-2048" or "EC-prime256v1"
	dds.ca.sn	The subject name of the Identity CA Certificate.
	dds.ca.algo	"RSA-2048" or "EC-prime256v1"

The value of the *class_id* shall be interpreted as composed of three parts: a *PluginClassName*, a *MajorVersion* and a *MinorVersion* according to the following format:

`<PluginClassName>:<MajorVersion>.<MinorVersion>`. The *PluginClassName* is separated from the *MajorVersion* by the last ':' character in the class_id. The *MajorVersion* and *MinorVersion* are separated by a '.' character. Accordingly this version of the specification has *PluginClassName* equal to "DDS:Auth:PKI-DH", *MajorVersion* set to 1, and *MinorVersion* set to 0.

9.3.2.2 DDS:Auth:PKI-DH IdentityStatusToken

The DDS:Auth:PKI-DH plugin shall set the attributes of the `IdentityStatusToken` object as specified in the table below:

Table 46 – AuthenticatedPeerCredentialToken class for the builtin Authentication plugin

Attribute name	Attribute value	
class_id	"DDS:Auth:PKI-DH:1.0"	
properties	*name*	*value*
(The presence of each of properties is optional)	ocsp_status	A DER-encoded OCSP response (using the ASN.1 type OCSPResponse defined in clause 4.2.1 of RFC 2560 [52]) that provides the status of the identity certificate of the DomainParticipant.

9.3.2.3 DDS:Auth:PKI-DH AuthenticatedPeerCredentialToken

The DDS:Auth:PKI-DH plugin shall set the attributes of the `AuthenticatedPeerCredentialToken` object as specified in the table below:

Table 47 – AuthenticatedPeerCredentialToken class for the builtin Authentication plugin

Attribute name	Attribute value	
class_id	"DDS:Auth:PKI-DH:1.0"	
properties	*name*	*value*
	c.id	Contents of the certificate signed by IdentityCA that was received from the peer DomainParticipant as part of the authentication process. Corresponds to the property with the same name received in the HandskaheRequestMessageToken or HandskaheReplyMessageToken.
	c.perm	Contents of the permissions document signed by the PermissionCA that that was received from the peer DomainParticipant as part of the authentication process. Corresponds to the property with the same name received in the HandskaheRequestMessageToken or HandskaheReplyMessageToken.

9.3.2.4 DDS:Auth:PKI-DH AuthRequestMessageToken

The DDS:Auth:PKI-DH plugin shall set the attributes of the `AuthRequestMessageToken` object as specified in the table below:

Table 48 – AuthRequestMessageToken class for the builtin Authentication plugin

Attribute name	Attribute value	
class_id	"DDS:Auth:PKI-DH:1.0+AuthReq"	
properties	*name*	*value*
	future_ challen ge	A 256-bit NONCE generated by the Participant, compliant with Section 8.6.7 of NIST Recommendation for Random Number Generation Using Deterministic Random Bit Generators [46]. The value shall match what will be sent on the ***challenge1*** property of the `HandshakeRequestMessageToken` or the ***challenge2*** property of the `HandshakeReplyMessageToken`.

9.3.2.5 DDS:Auth:PKI-DH HandshakeMessageToken

The DDS:Auth:PKI-DH plugin uses several `HandshakeMessageToken` object formats:

- `HandshakeRequestMessageToken` objects
- `HandshakeReplyMessageToken` objects
- `HandshakeFinalMessageToken` objects

9.3.2.5.1 HandshakeRequestMessageToken objects

The attributes in `HandshakeRequestMessageToken` objects shall be set as specified in the table below. References to the `DomainParticipant` within the table refer to the `DomainParticipant` that is creating the `HandshakeRequestMessageToken`.

Table 49 – HandshakeRequestMessageToken for the builtin Authentication plugin

Attribute name	Attribute value	
class_id	"DDS:Auth:PKI-DH:1.0+Req"	
binary_properties	**name**	**value**
	c.id	Contents of the certificate signed by IdentityCA that was configured using the Participant PropertyQosPolicy with name "dds.sec.auth.identity_certificate"
	c.perm	Contents of the permissions document signed by the PermissionCA that was configured using the Participant PropertyQosPolicy with name "dds.sec.access.permissions"
	c.pdata	The CDR Big Endian Serialization of the ParticipantBuiltinTopicData
	c.dsign_algo	Digital signature algorithm identifier. Either "RSASSA-PSS-SHA256" or "ECDSA-SHA256"

	c.kagree_algo	Key agreement algorithm identifier.
		Either "DH+MODP-2048-256" or "ECDH+prime256v1-CEUM"
	hash_c1	SHA-256 hash of the CDR Big Endian serialization of a BinaryPropertySeq object containing all the properties above that start with "c." placed in the same order as they appear above.
		Inclusion of the *hash_c1* property is optional. Its only purpose is to facilitate troubleshoot interoperability problems.
	dh1	The CDR Big Endian Serialization of a Diffie-Hellman Public Key chosen by the Participant. This will be used for key agreement.
	challenge1	A 256-bit NONCE generated by the Participant, compliant with Section 8.6.7 of NIST Recommendation for Random Number Generation Using Deterministic Random Bit Generators [46].
		If the `validate_remote_identity` returned a non-NIL `AuthRequestMessageToken`, then the value shall match what was sent on the `AuthRequestMessageToken` *future_challenge* property.
	ocsp_status	Inclusion of this property is optional.
		A DER-encoded OCSP response (using the ASN.1 type OCSPResponse defined in clause 4.2.1 of RFC 2560 [52]) that provides the status of the identity certificate in the **c.id** property.

Plugin implementations may add extra properties as long as the names comply with the rules defined in in 7.2.1. Plugin implementations shall ignore any properties they do not understand.

If the Participant Identity uses a RSA Public Key, then the *c.dsign_algo* shall be "RSASSA-PSS-SHA256".

If the Participant Identity uses a EC Public Key, then the *c.dsign_algo* shall be "ECDSA-SHA256".

9.3.2.5.2 HandshakeReplyMessageToken

The attributes in the `HandshakeReplyMessageToken` objects are set as specified in the table below. References to the `DomainParticipant` within the table refer to the `DomainParticipant` that is creating the `HandshakeReplyMessageToken`.

Table 50 – HandshakeReplyMessageToken for the builtin Authentication plugin

Attribute name	Attribute value

class_id	"DDS:Auth:PKI-DH:1.0+Reply"	
binary_properties	*name*	*value*
	c.id	Contents of the certificate signed by IdentityCA that was configured using the Participant PropertyQosPolicy with name "dds.sec.auth.identity_certificate"
	c.perm	Contents of the permissions document signed by the PermissionCA that was configured using the Participant PropertyQosPolicy with name "dds.sec.access.permissions"
	c.pdata	The CDR Big Endian Serialization of the ParticipantBuiltinTopicData
	c.dsign_algo	Digital signature algorithm identifier. Either "RSASSA-PSS-SHA256" or "ECDSA-SHA256"
	c.kagree_algo	Key agreement algorithm identifier. Either "DH+MODP-2048-256" or "ECDH+prime256v1-CEUM"
	hash_c2	SHA-256 hash of the CDR Big Endian serialization of a BinaryPropertySeq object containing all the properties above that start with "c." placed in the same order as they appear above. Inclusion of the *hash_c2* property is optional. Its only purpose is to facilitate troubleshoot interoperability problems.
	dh2	The CDR Big Endian Serialization of a Diffie-Hellman Public Key chosen by the Participant. This will be used to establish the shared secret.
	hash_c1	The value of the related HandshakeRequestMessageToken property hash_c1. Inclusion of the hash_c1 property is optional. Its only purpose is to facilitate troubleshoot interoperability problems.
	dh1	The value of the related HandshakeRequestMessageToken property dh1. Inclusion of the dh1 property is optional. Its only purpose is to facilitate troubleshoot interoperability problems.
	challenge1	Value of the related HandshakeRequestMessageToken property challenge1.

	challenge2	A 256-bit NONCE generated by the Participant, compliant with Section 8.6.7 of NIST Recommendation for Random Number Generation Using Deterministic Random Bit Generators [46]. If the `validate_remote_identity` returned a non-NIL `AuthRequestMessageToken`, then the value shall match what was sent on the `AuthRequestMessageToken` *future_challenge* property.
	ocsp_status	Inclusion of this property is optional. A DER-encoded OCSP response (using the ASN.1 type OCSPResponse defined in clause 4.2.1 of RFC 2560 [52]) that provides the status of the identity certificate in the **c.id** property.
	signature	The Digital Signature of the CDR Big Endian serialization of a BinaryPropertySeq object containing the properties: hash_c2, challenge2, dh2, challenge1, dh1, and hash_c1, placed in that order. All the aforementioned properties shall appear within the signature even if some of the optional properties do not appear separately as properties in the HandshakeReplyMessageToken.

Plugin implementations may add extra properties as long as the names comply with the rules defined in 7.4.3.5. Plugin implementations shall ignore any properties they do not understand.

If the value of the ***c. kagree_algo*** property is "DH+MODP-2048-256", then:

- The Diffie-Hellman Public Key shall be for the 2048-bit MODP Group with 256-bit Prime Order Subgroup, see IETF RFC 5114 [47], section 2.3.
- The Key Agreement Algorithm shall be the "dhEphem, C(2e, 0s, FFC DH) Scheme" defined in section 6.1.2.1 of NIST Special Publication 800-56A Revision 2 [48].

Non-normative note: The OpenSSL 1.0.2 operation DH_get_2048_256() retrieves the parameters for the 2048-bit MODP Group with 256-bit Prime Order Subgroup.

If the value of the ***c.kagree_algo*** property is "ECDH+prime256v1-CEUM", then:

- The Diffie-Hellman Public Key shall be for the NIST's EC Curve P-256 as defined in appendix D of FIPS 186-4 [42] also known as prime256v1 in ANSI X9.62-2005 [41].
- The Key Agreement Algorithm shall be the "(Cofactor) Ephemeral Unified Model, C(2e, 0s, ECC CDH)" defined in section 6.1.2.2 of NIST Special Publication 800-56A Revision 2 [48]. See also section 3.1 "Ephemeral Unified Model" of NIST Suite B Implementer's Guide to NIST SP 800-56A [49].

Regardless of the key agreement algorithm, the `SharedSecret` (see Table 53) shall be computed as the SHA256 hash of the derived shared secret computed by the key agreement algorithm. [Non-

normative: This is done to accommodate the use of cryptographic libraries that do not provide direct access to the derived shared secret and only allow retrieval of the SHA256 of the shared secret.]

The digital signature shall be computed using the Private Key associated with the DomainParticipant, which corresponds to the Public Key that appears in the Identity Certificate.

If the Participant Private Key is a RSA key, then:

- The value of the **c.dsign_algo** property shall be "RSASSA-PSS-SHA256".
- The digital signature shall be computed using the RSASSA-PSS algorithm specified in PKCS #1 (IETF 3447) RSA Cryptography Specifications Version 2.1 [44], using SHA256 as hash function, and MGF1 with SHA256 (mgf1sha256) as mask generation function.

If the Participant Private Key is an EC key, then:

- The value of the **c.dsign_algo** shall be "ECDSA-SHA256".
- The digital signature shall be computed using the ECDSA-SHA256 algorithm specified in ANSI X9.62-2005 [41].

9.3.2.5.3   HandshakeFinalMessageToken

`HandshakeFinalMessageToken` objects are used to finish an authentication handshake.

The attributes in the `HandshakeFinalMessageToken` objects shall be set as specified in the table below.

References to the `DomainParticipant` within the table refer to the `DomainParticipant` that is creating the `HandshakeFinalMessageToken`.

Table 51 – HandshakeFinalMessageToken for the builtin Authentication plugin

Attribute name	Attribute value	
class_id	"DDS:Auth:PKI-DH:1.0+Final".	
binary_properties	*name*	*value*
	hash_c1	The value of the related HandshakeRequestMessageToken property hash_c1. Inclusion of the hash_c1 property is optional. Its only purpose is to facilitate troubleshoot interoperability problems.
	hash_c2	The value of the related HandshakeReplyMessageToken property hash_c2. Inclusion of the hash_c2 property is optional. Its only purpose is to facilitate troubleshoot interoperability problems.
	dh1	The value of the related HandshakeRequestMessageToken property dh1. Inclusion of the dh1 property is optional. Its only purpose is to facilitate troubleshoot interoperability problems.

	dh2	The value of the related HandshakeReplyMessageToken property dh2.
		Inclusion of the dh2 property is optional. Its only purpose is to facilitate troubleshoot interoperability problems. |
	challenge1	Value of HandshakeRequestMessageToken property challenge1
	challenge2	Value of HandshakeReplyMessageToken property challenge2
	signature	The Digital Signature of the CDR Big Endian serialization of a BinaryPropertySeq object containing the properties: hash_c1, challenge1, dh1, challenge2, dh2, and hash_c2, placed in that order.
		All the aforementioned properties shall appear within the signature even if some of the optional properties do not appear separately as properties in the HandshakeFinalMessageToken.

The Diffie Hellman public key shall be for the same algorithm and Domain Parameters that were used for the `HandshakeRequestMessageToken` key received as value of the ***dh2*** property. The parameters and algorithm shall be determined based on the value of the `HandshakeRequestMessageToken` parameter with key ***c.kagree_algo***. In other words, it is the Participant that creates the `HandshakeRequestMessageToken` the one that controls the key agreement algorithm used.

The digital signature shall be computed using the Private Key associated with the DomainParticipant, which corresponds to the Public Key that appears in the Identity Certificate.

If the Participant Private Key is a RSA key, then the digital signature shall be computed using the RSASSA-PSS algorithm specified in PKCS #1 (IETF 3447) RSA Cryptography Specifications Version 2.1 [44], using SHA256 as hash function, and MGF1 with SHA256 (mgf1sha256) as mask generation function.

If the Participant Participant Private Key is an EC key, then the digital signature shall be computed using the ECDSA-SHA256 algorithm specified in ANSI X9.62-2005 [41].

9.3.3 DDS:Auth:PKI-DH plugin behavior

The table below describes the actions that the DDS:Auth:PKI-DH plugin performs when each of the plugin operations is invoked.

Table 52 – Actions undertaken by the operations of the builtin Authentication plugin

| `validate_local_identity` | This operation shall receive the ***participant_guid*** associated with the local `DomainParticipant` whose identity is being validated. |
| | The operation shall receive the `DomainParticipantQos` with a `PropertyQosPolicy` containing the properties defined in section 9.3.1. |

	The operation shall verify the validity of the X509 certificate associated with the property named *dds.sec.auth.identity_certificate* using the CA configured by the *dds.sec.auth.identity_ca* property.. The operation shall check a CRL and/or an OCSP (RFC 2560 [52]) responder. This includes checking the expiration date of the certificate.
	
	If the above check fails the operation shall return VALIDATION_FAILED.
	
	The operation shall fill the *handle* with an implementation-dependent reference that allows the implementation to retrieve at least the following information:
	
	    1. The private key associated with the *identity_credential*
	    2. The public key associated with the *identity_credential*
	    3. The *participant_guid*
	
	The operation shall return the 16-byte *adjusted_participant_guid* GUID consisting of the same EntityId_t and a GuidPrefix_t computed as follows:
	
	• The first bit (bit 0) shall be set to 1.
	• The 47 bits following the first bit (bits 1 to 47) shall be set to the 47 first bits of the SHA-256 hash of the ASN.1 DER encoding of the SubjectName [40] appearing on the *identity_credential*.
	• The following 48 bits (bits 48 to 95) shall be set to the first 48 bits of the SHA-256 hash of the *candidate_participant_guid*.
	
	If successful, the operation shall return VALIDATION_OK.

get_identity_token	The operation shall receive the *handle* corresponding to the one returned by a successful previous call to *validate_local_identity*. If the above condition is not met the operation shall return the exception DDS_SecurityException_PreconditionError. This operation shall return an IdentityToken object with the content specified in 9.3.2.1.
set_permissions_cr edential_and_token	This operation shall store the PermissionsCredentialToken and the PermissionsToken internally to the plugin and associate them with the DomainParticipant represented by the IdentityHandle.
validate_remote_id entity	The operation shall receive the IdentityToken of the remote participant in the argument *remote_identity_token*. The contents of the IdentityToken shall be identical to what would be returned by a call to get_identity_token on the

	Authentication plugin of the remote DomainParticipant associated with the *remote_participant_guid*.
	The operation shall compare the *class_id* of the *local identity_token* with that of the *remote_identity_token*. If the P*luginClassName* or the *MajorVersion* are different, it shall return VALIDATION_FAILED.
	If the *remote_auth_request_token* is NIL, the operation shall generate a *local_auth_request_token* AuthRequestMessageToken (see 9.3.2.4), otherwise the *local_auth_request_token* shall be set to TokenNIL. Note that a *local_auth_request_token* is returned as an out parameter.
	The operation shall compare lexicographically the *remote_participant_guid* with the participant key obtained from the *local_identity_handle*.
	If the *remote_participant_guid* > *local_participant_guid*, the operation shall return VALIDATION_PENDING_HANDSHAKE_REQUEST.
	If the *remote_participant_guid* < *local_participant_guid*, the operation shall return VALIDATION_PENDING_HANDSHAKE_MESSAGE.
	In both scenarios the *remote_identity_handle* shall be filled with a reference to internal plugin information that identifies the remote participant and associates it to the contents of the *remote_identity_token,* the *local_auth_request_token*, the *remote_auth_request_token* and any additional information required for the challenge protocol.
begin_handshake_re quest	The operation shall receive the *initiator_identity_handle* corresponding to the *local_identity_handle* of a previous invocation to the *validate_remote_identity* operation that returned VALIDATION_PENDING_HANDSHAKE_REQUEST.
	The operation shall also receive the *replier_identity_handle* corresponding to the *remote_identity_handle* returned by that same invocation to the *validate_remote_identity* operation.
	The operation shall also receive the *serialized_local_participant_data* associated with the local DomainParticipant. This will be used to set the value of the property named "c.pdata".
	The operation shall return the *handshake_message* containing a HandshakeRequestMessageToken object with contents as defined in 9.3.2.5.1.
	The operation shall check the content of the

	local_auth_request_token associated with the *remote_identity_handle*. If the token was different from `TokenNIL`, the operation shall use the value of property named "`future_challenge`" found in the *local_auth_request_token* to fill the property named "`challenge1`" of the *handshake_message* returned.
	
	The operation shall fill the *handshake_handle* with an implementation-dependent reference that allows the implementation to retrieve at least the following information:
	
	    1. The *local_identity_handle*
	    2. The *remote_identity_handle*
	    3. The value attribute of the *handshake_message* returned
	
	The operation shall return VALIDATION_PENDING_HANDSHAKE_MESSAGE.

`begin_handshake_reply`	The operation shall receive the *replier_identity_handle* corresponding to *local_identity_handle* of a previous invocation to the *validate_remote_identity* operation that returned VALIDATION_PENDING_HANDSHAKE_MESSAGE. The operation shall also receive the *initiator_identity_handle* corresponding to the *remote_identity_handle* returned by that same invocation to the `validate_remote_identity` operation. The operation shall also receive the *serialized_local_participant_data* associated with the local `DomainParticipant`. This will be used to set the value of the property named "c.pdata". If any of the above conditions is not met, the operation shall return the exception DDS_SecurityException_PreconditionError. The operation shall check the content of the *remote_auth_request_token* associated with the *remote_identity_handle*. If the token was different from `TokenNIL`, the operation shall verify that the property named "`future_challenge`" found in that token is the same value as the property named "`challenge1`" found in the *handshake_message_in* `HandshakeRequestMessageToken`. If the condition is not met, the operation shall return VALIDATION_FAILED. The operation shall check the content of the *local_auth_request_token* associated with the *remote_identity_handle*. If the token was different from `TokenNIL`, the operation shall use the value of property named "`future_challenge`" found in the

local_auth_request_token to fill the property named "challenge2″ of the *handshake_message* returned.

The operation shall verify the validity of the IdentityCredential contained in the property named "c.id" found in the *handshake_message_in* HandshakeRequestMessageToken. This verification shall be done using the locally configured CA in the same manner as the validate_local_identity operation.

If the *handshake_message_in* does not contain the aforementioned property or the verification fails, then the operation shall fail and return ValidationResult_Fail.

If the property **ocsp_status** is present, the operation shall verify that the OCSP response included in the property corresponds to the identity in the **c.id** property. The operation shall use the OCSP response to verify the status of the IdentityCredential. If that status is good and the validity interval has not been exceeded it shall accept that as proof that the IdentityCredential is still valid. If the status is revoked, the operation shall fail and return ValidationResult_Fail. If the status is different from the aforementioned ones it shall behave as if the **ocsp_status** property was not present.

If the property **ocsp_status** is not present, the operation shall use its own means to determine the status of the IdentityCredential. This may performing an OCSP query or consulting a CRL list. The specific behavior is implementation specific.

The operation shall verify that the first bit of the *participant_guid* of the ParticipantBuiltinTopic data inside the "c.pdata" is set to 1 and that the following 47 bits match the first 47 bits of the SHA-256 hash of the SubjectName appearing in the IdentityCredential. If this verification fails, the operation shall fail and return ValidationResult_Fail.

The operation shall fill the *handshake_message_out* with a HandshakeReplyMessageToken object with the content specified in 9.3.2.5.2.

The operation shall fill the *handshake_handle* with an implementation-dependent reference that allows the implementation to retrieve at least the following information:

1. The *replier_identity_handle*
2. The *initiator_identity_handle*
3. The value attribute of the *challenge_message* returned
4. The property with name "dds.sec.permissions" found within the *handshake_message_in* if present

	The operation shall return VALIDATION_PENDING_HANDSHAKE_MESSAGE.

`process_handshake`
 on a ***handshake_handle***
 created by
 `begin_handshake_re`
 `quest`
	The operation shall be called with the ***handshake_handle*** returned by a previous call to ***begin_handshake_request*** that returned VALIDATION_PENDING_HANDSHAKE_MESSAGE.
	
	 The ***handshake_message_in*** shall correspond to a `HandshakeReplyMessageToken` object received as a reply to the ***handshake_message*** `HandshakeRequestMessageToken` object associated with the ***handshake_handle***.
	
	 If any of the above conditons is not met, the operation shall return the exception DDS_SecurityException_PreconditionError.
	
	 The operation shall verify that the contents of the ***handshake_message_in*** correspond to a `HandshakeReplyMessageToken` as described in 9.3.2.5.2.
	
	 The operation shall check the content of the ***remote_auth_request_token*** associated with the ***remote_identity_handle***. If the token was different from `TokenNIL`, the operation shall verify that the property named "`future_challenge`" found in that token is the same value as the property named "`challenge2`" found in the ***handshake_message_in*** `HandshakeReplyMessageToken`. If the condition is not met, the operation shall return VALIDATION_FAILED.
	
	 The operation shall verify the validity of the `IdentityCredential` contained in the property named "`c.id`" found in the ***handshake_message_in*** `HandshakeReplyMessageToken`. This verification shall be done using the locally configured CA in the same manner as the ***validate_local_identity*** operation.
	
	 If the ***handshake_message_in*** does not contain the aforementioned property or the verification fails, then the operation shall fail and return ValidationResult_Fail.
	
	 If the property **ocsp_status** is present, the operation shall verify that the OCSP response included in the property corresponds to the identity in the **c.id** property. The operation shall use the OCSP response to verify the status of the `IdentityCredential`. If that status is good and the validity interval has not been exceeded it shall accept that as proof that the IdentityCredential is still valid. If the status is revoked, the operation shall fail and return

	ValidationResult_Fail. If the status is different from the aforementioned ones it shall behave as if the **ocsp_status** property was not present.
	If the property **ocsp_status** is not present, the operation shall use its own means to determine the status of the `IdentityCredential`. This may performing an OCSP query or consulting a CRL list. The specific behavior is implementation specific.
	The operation shall check that the challenge1 matches the one that was sent on the `HandshakeRequestMessageToken.`
	The operation shall validate the digital signature in the "`signature`" property, according to the algorithm described in section 9.3.2.5.2.
	If the specified checks do not succeed, the operation shall return VALIDATION_FAILED.
	The operation shall create a `HandshakeFinalMessageToken` object as described in 9.3.2.5.3. The operation shall fill the **handshake_message_out** with the created `HandshakeFinalMessageToken` object.
	The operation shall store the **value** of property with **name** "`dds.sec.`" found within the **handshake_message_in**, if present and associate it with the **handshake_handle** as the `PermissionsCertificate` of remote `DomainParticipant`.
	The operation shall use the Diffie Hellman Public Key in the "`dh2`" property in combination with the Diffie Hellman Private Key it used to compute the `HandshakeFinalMessageToken` "`dh1`" property to compute the shared secret. The algorithm shall be as described in section 9.3.2.5.2.
	On success the operation shall return VALIDATION_OK_FINAL_MESSAGE.
`process_handshake` on a **handshake_handle** created by `begin_handshake_reply`	The operation shall be called with the **handshake_handle** returned by a previous call to **begin_handshake_reply** that returned VALIDATION_PENDING_HANDSHAKE_MESSAGE.
	The **handshake_message_in** shall correspond to the one received as a reply to the **handshake_message_out** associated with the **handshake_handle**.
	If any of the above conditions is not met, the operation shall return the exception DDS_SecurityException_PreconditionError.

	The operation shall verify that the contents of the *handshake_message_in* correspond to a `HandshakeFinalMessageToken` object as described in 9.3.2.5.3.
	The operation shall check that the challenge1 and challenge2 match the ones that were sent on the `HandshakeReplyMessageToken.`
	The operation shall validate the digital signature in the "`signature`" property, according to the expected contents and algorithm described in section 9.3.2.5.3.
	The operation shall use the Diffie Hellman Public Key in the "`dh1`" property in combination with the Diffie Hellman Private Key it used to compute the `HandshakeReplyMessageToken` "`dh2`" property to compute the shared secret. The algorithm shall be as described in section 9.3.2.5.2.
	On success the operation shall return VALIDATION_OK.
`get_shared_secret`	This operation shall be called with the *handshake_handle* that was previously used to call either *process_handshake* and for which the aforementioned operation returned VALIDATION_OK_FINAL_MESSAGE or VALIDATION_OK.
	If the above conditon is not met, the operation shall return the exception DDS_SecurityException_PreconditionError.
	The operation shall return a `SharedSecretHandle` that is internally associated with the SharedSecret established as part of the handshake.
	On failure the operation shall return nil.
`get_authenticated_` `peer_credential_to` `ken`	This operation shall be called with the *handshake_handle* that was previously used to call either *process_handshake* and for which the aforementioned operation returned VALIDATION_OK_FINAL_MESSAGE or VALIDATION_OK.
	If the above conditon is not met, the operation shall return the exception DDS_SecurityException_PreconditionError.
	The operation shall return the `AuthenticatedPeerCredentialToken` of the peer `DomainParticipant` associated with the *handshake_handle*. If the `DomainParticipant` initiated the handshake, then the peer `AuthenticatedPeerCredentialToken` is constructed from the `HandshakeReplyMessageToken`, otherwise it is constructed from the `HandshakeRequestMessageToken`. See section 0.

	On failure the operation shall return nil.
`set_listener`	This operation shall save a reference to the listener object and associate it with the specified `IdentityHandle`.
`return_identity_to ken`	This operation shall behave as specified in 8.3.2.11.13.
`return_peer_permis sions_credential_t oken`	This operation shall behave as specified in 8.3.2.11.15.
`return_handshake_h andle`	This operation shall behave as specified in 8.3.2.11.16.
`return_identity_ha ndle`	This operation shall behave as specified in 8.3.2.11.17.
`return_sharedsecre t_handle`	This operation shall behave as specified in 8.3.2.11.18.

9.3.4 DDS:Auth:PKI-DH plugin authentication protocol

The operations the Secure DDS implementation executes on the `Authentication` plugin combined with the behavior of the DDS:Auth:PKI-DH result in an efficient 3-message protocol that performs mutual authentication and establishes a shared secret.

The rest of this sub clause describes the resulting protocol.

The authentication protocol is symmetric, that is there are no client and server roles. But only one `DomainParticipant` should initiate the protocol. To determine which of the two `DomainParticipant` entities shall initiate the protocol, each `DomainParticipant` compares its own GUID with that of the other DomainParticipant. The `DomainParticipant` with the lower GUID (using lexicographical order) initiates the protocol.

9.3.4.1 Terms and notation

The table below summarizes the terms used in the description of the protocol.

Table 53 – Terms used in the description of the builtin authentication protocol

Term	Meaning
Participant1	The `DomainParticipant` that initiates the handshake protocol.
	It calls `begin_handshake_request`, sends the `HandshakeRequestMessageToken`, receives the `HandshakeReplyMessageToken`, and sends the `HandshakeFinalMessageToken`).

Participant2	The DomainParticipant that does not initiate the handshake protocol.
	It calls `begin_handshake_reply`, receives the `HandshakeRequestMessageToken`, sends the `HandshakeReplyMessageToken`, and receives the `HandshakeFinalMessageToken`).
PubK_1	The Public Key of Participant1.
PubK_2	The Public Key of Participant2.
PrivK_1	The Private Key of Participant1.
PrivK_2	The Private Key of Participant2.
Cert1	The `IdentityCertificate` (signed by the shared CA) of Participant A. It contains PubK_1.
Cert2	The `IdentityCertificate` (signed by the shared CA) of Participant 2. It contains PubK_2.
Perm1	Permissions document of Participant1 (signed by Permissions CA).
Perm2	Permissions document of Participant2 (signed by Permissions CA).
Pdata1	ParticipantBuiltinTopicData of Participant1.
Pdata2	ParticipantBuiltinTopicData of Participant2.
Dsign_algo1	Token identifying the Digital Signature Algorithm for Participant1.
Dsign_algo2	Token identifying the Digital Signature Algorithm for Participant2.
Kagree_algo1	Token identifying the Key Agreement Algorithm selected by Participant1 that shall be used to establish the shared secret.
Kagree_algo2	Token identifying the Key Agreement Algorithm used by Participant2. It shall be set to match the one received from Participant1 in Kagree_algo1and used to establish the shared secret.
Challenge1	The challenge created by Participant1.
Challenge2	The challenge created by Participant2.
DH1	Diffie-Hellman Public Key generated by Participant1.
DH2	Diffie-Hellman Public Key generated by Participant2.
DHSharedSecret	The shared secret computed combining DH1 and DH2 with the DH secret key each participant has.
SharedSecret	The SHA256 Hash of the DHSharedSecret.
C1	A shortcut for the list: Cert1, Perm1, Pdata1, Dsign_algo1, Kagree_algo1.
C2	A shortcut for the list: Cert2, Perm2, Pdata2, Dsign_algo2, Kagree_algo2.

The table below summarizes the notation and transformation functions used in the description of the protocol:

Table 54 – Notation of the operations/transformations used in the description of the builtin authentication protocol

Function / notation	*meaning*
Sign(data)	Signs the 'data' argument using the Participant Private Key.
Hash(data)	Hashes the 'data' argument using SHA-256.
data1 | data2	The symbol '|' is used to indicate byte concatenation.

9.3.4.2 Protocol description

The table below describes the resulting 3-way protocol that establishes authentication and a shared secret between Participant_A and Participant_B.

Table 55 – Description of built-in authentication protocol

Participant A	Participant B

Is configured with PrivK_1 and C1 where
 	
C1 = Cert1, Perm1, Pdata1, Dsign_algo1, Kagree_algo1
 	
Generates a random Challenge1.
 	
Generates DH1.
 	
Sends:
 	
`HandshakeRequestMessageToken:`
`(C1, Hash(C1), Challenge1, DH1)`
 	
Note: In the above message Hash(C1) may be omitted.
	Is configured with PrivK_2 and C2 where
	
	C2 = Cert2, Perm2, Pdata2, Dsign_algo2, Kagree_algo2

	Receives `HandshakeRequestMessageToken` Verifies Cert1 with the configured Identity CA Verifies Hash(C1) Generates a random Challenge2 Generates DH2 Sends: `HandshakeReplyMessageToken:` `(C2, Hash(C2),` `    Challenge1, Challenge2,` `    DH2, Hash(C1), DH1,` ` Sign(Hash(C2)	Challenge2`  `	DH2	Challenge1	DH1`  `	Hash(C1)))` Note: In the above message Hash(C2) , Hash(C1) and DH1 may be omitted outside the signature.

Receives `HandshakeReplyMessageToken` Verifies Cert2 with the configured Identity CA Verifies signature against PubK2 Computes shared secret from DH2 and the DH private key used for DH1 Sends: `HandshakeFinalMessageToken:` `( Hash(C1), Hash(C2), DH1, DH2,` `Challenge1, Challenge2,` `Sign(Hash(C1)	Challenge1	DH1`         `| Challenge2	DH2`         `| Hash(C2)))`	
Note: In the above message Hash(C1) , Hash(C2), DH1, and DH2 may be omitted outside the signature.	Receives `HandshakeFinalMessageToken` Checks Hash(C1) matches the `HandshakeRequestMessageToken` Verifies the signature in `HandshakeFinalMessageToken` against PubK_1 Computes shared secret from DH1 and the DH private key used for DH2

9.4 Builtin Access Control: DDS:Access:Permissions

This builtin `AccessControl` plugin is referred to as the "DDS:Access:Permissions" plugin.

The DDS:Access:Permissions implements the `AccessControl` plugin API using a permissions document signed by a shared Certificate Authority (CA).

The shared CA could be an existing one (including the same CA used for the `Authentication` plugin), or a new one could be created for the purpose of assigning permissions to the applications on a DDS Domain. The nature or manner in which the CA is selected is not important because the way it is used enforces a shared recognition by all participating applications.

Each `DomainParticipant` has an associated instance of the DDS:Access:Permissions plugin.

9.4.1 Configuration

The DDS:Access:Permissions plugin is configured with three documents:

- The Permissions CA certificate
- The Domain governance signed by the Permissions CA
- The DomainParticipant permissions signed by the Permissions CA

The configuration of the builtin access control plugin shall be done using the `PropertyQosPolicy` of the `DomainParticipantQos`. The specific properties used are described in Table 56 below.

Table 56 – Properties used to configure the builtin AccessControl plugin

Property Name *(all properties have "dds.sec.access" prefix)*	*Property Value* *(all these properties shall have propagate set to FALSE)* *URI syntax follows IETF RFC 3986.* *URI "data" schema follows IETF RFC 2397* *Vendors may support additional schemas*
permissions_ca	URI to a X509 certificate for the PermissionsCA in PEM format. Supported URI schemes: file, data, pkcs11 The **file** and **data** schemas shall refer to a X.509 v3 certificate (see X.509 v3 ITU-T Recommendation X.509 (2005) [39]) in PEM format. Examples: file:permissions_ca.pem file:/home/myuser/ permissions_ca.pem data:,-----BEGIN CERTIFICATE----- MIIC3DCCAcQCCQCWE5x+Z ... PhovK0mp2ohhRLYI0ZiyYQ== -----END CERTIFICATE----- pkcs11:object= MyPermissionsCACert;type=cert

governance	URI to the shared Governance Document signed by the Permissions CA in S/MIME format
	URI schemes: file, data
	
	Example file URIs:
	file:governance.smime
	file:/home/myuser/governance.smime
	
	Example data URI:
	data:,MIME-Version: 1.0
	Content-Type: multipart/signed; protocol="application/x-pkcs7-signature"; micalg="sha-256"; boundary="----F9A8A198D6F08E1285A292ADF14DD04F"
	
	This is an S/MIME signed message
	
	------F9A8A198D6F08E1285A292ADF14DD04F
	<?xml version="1.0" encoding="UTF-8"?>
	<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:noNamespaceSchemaLocation="omg_shared_ca_governance.xsd">
	  <domain_access_rules>
	…
	  </domain_access_rules>
	</dds>
	…

	------F9A8A198D6F08E1285A292ADF14DD04F Content-Type: application/x-pkcs7-signature; name="smime.p7s" Content-Transfer-Encoding: base64 Content-Disposition: attachment; filename="smime.p7s" MIIDuAYJKoZIhv ...al5s= ------F9A8A198D6F08E1285A292ADF14DD04F—
permissions	URI to the DomainParticipant permissions document signed by the Permissions CA in S/MIME format URI schemes: file, data Example file URIs: file:participant1_permissions.smime file:/home/myuser/participant1_permissions.smime

9.4.1.1 Permissions CA Certificate

This is a X.509 certificate that contains the Public Key of the CA that will be used to sign the Domain Governance and Domain Permissions document. The certificate can be self-signed or signed by some other CA. Regardless of this the Public Key in the Certificate shall be trusted to sign the aforementioned Governance and Permissions documents (see 9.4.1.2 and 9.4.1.3).

The Permissions CA Certificate shall be provided to the plugins using the `PropertyQosPolicy` on the `DomainParticipantQos` as specified in Table 56.

9.4.1.2 Domain Governance Document

The domain governance document is an XML document that specifies how the domain should be secured.

The domain governance document shall be signed by the Permissions CA. The signed document shall use S/MIME version 3.2 format as defined in IETF RFC 5761 using SignedData Content Type (section 2.4.2 of IETF RFC 5761) formatted as multipart/signed (section 3.4.3 of IETF RFC 5761). This corresponds to the mime-type application/pkcs7-signature. Additionally the signer certificate shall be included within the signature.

The signed governance document shall be provided to the plugins using the `PropertyQosPolicy` on the `DomainParticipantQos` as specified in Table 56.

The governance document specifies which DDS domain IDs shall be protected and the details of the protection. Specifically this document configures the following aspects that apply to the whole domain:

- Whether the discovery information should be protected and the kind of protection: only message authentication codes (MACs) or encryption followed by MAC.
- Whether the whole RTPS message should be protected and the kind of protection. This is in addition to any protection that may occur for individual submessages and for submessage data payloads.
- Whether the liveliness messages should be protected.
- Whether a discovered DomainParticipant that cannot authenticate or fail the authentication should be allowed to join the domain and see any discovery data that are configured as 'unprotected' and any Topics that are configured as 'unprotected'.

- Whether any discovered DomainParticipant that authenticates successfully should be allowed to join the domain and see the discovery data without checking the access control policies.

In addition, the domain governance document specifies how the information on specific Topics within the domain should be treated. Specifically:

- Whether the discovery information on specific Topics should be sent using the secure (protected) discovery writers or using the regular (unprotected) discovery writers.
- Whether read access to the Topic should be open to all or restricted to the DomainParticipants that have the proper permissions.
- Whether write access to the Topic should be open to all or restricted to the DomainParticipants that have the proper permissions.

- Whether the metadata information sent on the Topic (sequence numbers, heartbeats, key hashes, gaps, acknowledgment messages, etc.) should be protected and the kind of protection (MAC or Encrypt then MAC).
- Whether the payload data sent on the Topic (serialized application level data) should be protected and the kind of protection (MAC or Encrypt then MAC).

9.4.1.2.1 Basic Protection Kinds

The domain governance document provides a means for the application to configure the kinds of cryptographic transformation applied to the complete RTPS Message, certain RTPS SubMessages, and the SerializedPayload RTPS submessage element that appears within the `Data` and `DataFrag` submessages.

The configuration allows specification of three protection levels: NONE, SIGN, ENCRYPT.

NONE indicates no cryptographic transformation is applied.

SIGN indicates the cryptographic transformation shall be purely a message authentication code (MAC), that is, no encryption is performed. Therefore the resulting `CryptoTransformIdentifier` for the output of the "encode" transformations shall have the *transformation_kind* attribute set to the `CRYPTO_TRANSFORMATION_KIND` variants `AES_128_GMAC` or `AES_256_GMAC`.

ENCRYPT indicates the cryptographic transformation shall be an encryption followed by a message authentication code (MAC) computed on the ciphertext, also known as Encrypt-then-MAC. Therefore the resulting `CryptoTransformIdentifier` for the output of the "encode" transformations shall have the *transformation_kind* attribute set to the `CRYPTO_TRANSFORMATION_KIND` variants `AES_128_GCM` or `AES_256_GCM`.

9.4.1.2.2 Protection Kinds

This configuration allows specification of two protection levels beyond the ones provided by the Basic Protection Kind (9.4.1.2.1): SIGN_WITH_ORIGIN_AUTHENTICATION and ENCRYPT_WITH_ORIGIN_AUTHENTICATION.

SIGN_WITH_ORIGIN_AUTHENTICATION indicates the cryptographic transformation shall be purely a set of message authentication codes (MAC), that is, no encryption is performed. This cryptographic transformation shall create a first "common authenticationcode" similar to the case where Protection Kind is SIGN. In addition the cryptographic transformation shall create additional authentication codes, each produced with a different secret key. Each of these additional secret keys

shall be shared only with a subset of the receivers. In the limit case each secret key is shared with only one receiver. The additional MACs prove to the receiver that the sender originated the message, preventing other receivers from impersonating the sender.

The resulting `CryptoTransformIdentifier` for the output of the "encode" transformations shall have the ***transformation_kind*** attribute set to the `CRYPTO_TRANSFORMATION_KIND` variants `AES_128_GMAC` or `AES_256_GMAC`.

ENCRYPT_WITH_ORIGIN_AUTHENTICATION indicates the cryptographic transformation shall be an encryption followed by a message authentication code (MAC) computed on the ciphertext, followed by additional authentication codes, Each of the additional authentication codes shall use a different secret key. The encryption and first (common) authentication code is similar to ones produced when the Protection Kind is ENCRYPT. The additional authentication codes are similar to the ones produced when the Protection Kind is **SIGN_WITH_ORIGIN_AUTHENTICATION**.

The resulting `CryptoTransformIdentifier` for the output of the "encode" transformations shall have the ***transformation_kind*** attribute set to the `CRYPTO_TRANSFORMATION_KIND` variants `AES_128_GCM` or `AES_256_GCM`.

9.4.1.2.3   Domain Governance document format

The format of this document defined using the following XSD:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xs:element name="dds" type="DomainAccessRulesNode" />

    <xs:complexType name="DomainAccessRulesNode">
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="domain_access_rules"
                        type="DomainAccessRules" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="DomainAccessRules">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="domain_rule" type="DomainRule" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="DomainRule">
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="domains" type="DomainIdSet" />
            <xs:element name="allow_unauthenticated_participants"
                        type="xs:boolean" />
            <xs:element name="enable_join_access_control"
                        type="xs:boolean" />
            <xs:element name="discovery_protection_kind"
                        type="ProtectionKind" />
```

```xml
            <xs:element name="liveliness_protection_kind"
                        type="ProtectionKind" />
            <xs:element name="rtps_protection_kind"
                        type="ProtectionKind" />
            <xs:element name="topic_access_rules"
                        type="TopicAccessRules" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="DomainIdSet">
        <xs:choice minOccurs="1" maxOccurs="unbounded">
            <xs:element name="id" type="DomainId" />
            <xs:element name="id_range" type="DomainIdRange" />
        </xs:choice>
    </xs:complexType>

    <xs:simpleType name="DomainId">
        <xs:restriction base="xs:nonNegativeInteger" />
    </xs:simpleType>

    <xs:complexType name="DomainIdRange">
        <xs:choice>
            <xs:sequence/>
                <xs:element name="min" type="DomainId" />
                <xs:element name="max" type="DomainId" minOccurs="0" />
            </xs:sequence/>
            <xs:element name="max" type="DomainId" />
        </xs:choice>
    </xs:complexType>

    <!-- DDSSEC11-11 -->
    <xs:simpleType name="ProtectionKind">
        <xs:restriction base="xs:string">
            <xs:enumeration value="ENCRYPT_WITH_ORIGIN_AUTHENTICATION" />
            <xs:enumeration value="SIGN_WITH_ORIGIN_AUTHENTICATION" />
            <xs:enumeration value="ENCRYPT" />
            <xs:enumeration value="SIGN" />
            <xs:enumeration value="NONE" />
        </xs:restriction>
    </xs:simpleType>

    <!-- DDSSEC11-11 -->
    <xs:simpleType name="BasicProtectionKind">
        <xs:restriction base="ProtectionKind">
            <xs:enumeration value="ENCRYPT" />
            <xs:enumeration value="SIGN" />
            <xs:enumeration value="NONE" />
        </xs:restriction>
    </xs:simpleType>

    <!-- DDSSEC-130 -->
    <xs:complexType name="TopicAccessRules">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="topic_rule" type="TopicRule" />
```

```
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="TopicRule">
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="topic_expression" type="TopicExpression" />
            <xs:element name="enable_discovery_protection"
                        type="xs:boolean" />
            <!-- DDSSEC11-85 -->
            <xs:element name="enable_liveliness_protection"
                        type="xs:boolean" />
            <xs:element name="enable_read_access_control"
                        type="xs:boolean" />
            <xs:element name="enable_write_access_control"
                        type="xs:boolean" />
            <xs:element name="metadata_protection_kind"
                        type="ProtectionKind" />
            <!-- DDSSEC11-11 -->
            <xs:element name="data_protection_kind"
                        type="BasicProtectionKind" />
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="TopicExpression">
        <xs:restriction base="xs:string" />
    </xs:simpleType>

</xs:schema>
```

9.4.1.2.4   Domain Access Rules Section

The XML domain governance document is delimited by the `<dds>` XML element tag and contains a single domain access rules Section delimited by the `<domain_access_rules>` XML element tag.

The domain access rules Section contains a set of domain rules each delimited by the `<domain_rule>` XML element tag.

9.4.1.2.5   Domain Rules

Each domain rule appears within the domain access rules Section delimited by the `<domain_rule>` XML element tag.

Each domain rule contains the following elements and sections:

1. Domains element

2. Allow Unauthenticated Participants element

3. Enable Join Access Control element

4. Discovery Protection Kind element

5. Liveliness Protection Kind element

6. RTPS Protection Kind element

7. Topic Access Rules Section

The contents and delimiters of each Section are described below.

The domain rules shall be evaluated in the same order as they appear in the document. A rule only applies to a particular `DomainParticipant` if the domain Section matches the DDS domain_id to which the `DomainParticipant` belongs. If multiple rules match, the first rule that matches is the only one that applies.

9.4.1.2.5.1 Domains element

This element is delimited by the XML element **<domains>**.

The value in this element identifies the collection of DDS `domain_id` values to which the rule applies.

The **<domains>** element can contain a single domain ID, for example:
```
<domains>
    <id>0</id>
</domains>
```

Or it can contain a range of domain IDs, for example:
```
<domains>
    <id_range>
        <min>10</min>
        <max>20</max>
    </id_range>
</domains>
```

Or it can contain a list of domain IDs and domain ID ranges, for example:
```
<domains>
    <id>0</id>
    <id_range>
        <min>10</min>
        <max>20</max>
    </id_range>
    <id>25</id>
    <id>27</id>
    <id_range>
        <min>40</min>
        <max>55</max>
    </id_range>
</domains>
```

9.4.1.2.5.2 Allow Unauthenticated Participants element

This element is delimited by the XML element **<allow_unauthenticated_participants>**.

This element may take the binary values TRUE or FALSE.

If the value is set to FALSE, the `ParticipantSecurityAttributes` returned by the `get_participant_sec_attributes` operation on the `AccessControl` shall have the ***allow_unauthenticated_participants*** member set to FALSE.

If the value is set to TRUE, the `ParticipantSecurityAttributes` returned by the `get_participant_sec_attributes` operation on the `AccessControl` shall have the ***allow_unauthenticated_participants*** member set to TRUE.

9.4.1.2.5.3   Enable Join Access Control element

This element is delimited by the XML element **<enable_join_access_control>**.

This element may take the binary values TRUE or FALSE.

If the value is set to FALSE, the `ParticipantSecurityAttributes` returned by the `get_participant_sec_attributes` operation on the `AccessControl` shall have the *is_access_protected* member set to FALSE.

If the value is set to TRUE, the `ParticipantSecurityAttributes` returned by the `get_participant_sec_attributes` operation on the `AccessControl` shall have the *is_access_protected* member set to TRUE.

9.4.1.2.5.4   Discovery Protection Kind element

This element is delimited by the XML element **<discovery_protection_kind>**.

The discovery protection element specifies the protection kind (see 9.4.1.2.2) used for the secure builtin `DataWriter` and `DataReader` entities used for discovery:

SPDPbuiltinParticipantsSecureWriter, SEDPbuiltinPublicationsSecureWriter, *SEDPbuiltinSubscriptionsSecureWriter, SPDPbuiltinParticipantsSecureReader, SEDPbuiltinPublicationsSecureReader*, *SEDPbuiltinSubscriptionsSecureReader*.

The discovery protection kind element may take five possible values: NONE, SIGN, ENCRYPT, SIGN_WITH_ORIGIN_AUTHENTICATION, or ENCRYPT_WITH_ORIGIN_AUTHENTICATION. The resulting behavior for the aforementioned builtin discovery secure entities shall be as specified in 9.4.1.2.2 with regards to the RTPS SubMessages.

This setting controls the contents of the `ParticipantSecurityAttributes` and `PluginParticipantSecurityAttributes` returned by the `AccessControl::get_participant_sec_attributes` operation on the `DomainParticipant`. Specifically:

- The attribute *is_discovery_protected* attribute in the `ParticipantSecurityAttributes` shall be set to FALSE if the value specified in the **<discovery_protection_kind>** element is NONE and to TRUE otherwise.
- The attribute *is_discovery_encrypted* in the `PluginParticipantSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **<discovery_protection_kind>** is ENCRYPT or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.
- The attribute *is_discovery_origin_authenticated* in the `PluginParticipantSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **<discovery_protection_kind>** is SIGN_WITH_ORIGIN_AUTHENTICATION or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.

9.4.1.2.5.5   Liveliness Protection Kind element

This element is delimited by the XML element **<liveliness_protection_kind>**.

The liveliness protection element specifies the protection kind (see 9.4.1.2.2) used for builtin `DataWriter` and `DataReader` associated with the ***ParticipantMessageSecure*** builtin Topic (see 7.4.2): ***BuiltinParticipantMessageSecureWriter*** and ***BuiltinParticipantMessageSecureReader.***

The liveliness protection kind element may five three possible values: NONE, SIGN, ENCRYPT, SIGN_WITH_ORIGIN_AUTHENTICATION, or ENCRYPT_WITH_ORIGIN_AUTHENTICATION.

This setting controls the contents of the `ParticipantSecurityAttributes` and `PluginParticipantSecurityAttributes` returned by the `AccessControl::get_participant_sec_attributes` operation on the `DomainParticipant`. Specifically:

- The attribute *is_liveliness_protected* in the `ParticipantSecurityAttributes` shall be set to FALSE if the value specified in the **<liveliness_protection_kind>** element is NONE and to TRUE otherwise.
- The attribute *is_liveliness_encrypted* in the `PluginParticipantSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **< liveliness_protection_kind>** is ENCRYPT or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.
- The attribute *is_liveliness_origin_authenticated* in the `PluginParticipantSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **< liveliness_protection_kind>** is SIGN_WITH_ORIGIN_AUTHENTICATION or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.

9.4.1.2.5.6   RTPS Protection Kind element

This element is delimited by the XML element **<rtps_protection_kind>**.

The RTPS protection kind element specifies the protection kind (see 9.4.1.2.2) used for the whole RTPS message.

The RTPS protection kind element may take five possible values: NONE, SIGN, ENCRYPT, SIGN_WITH_ORIGIN_AUTHENTICATION, or ENCRYPT_WITH_ORIGIN_AUTHENTICATION. The resulting behavior for the RTPS message cryptographic transformation shall be as specified in 9.4.1.2.2.

This setting controls the contents of the `ParticipantSecurityAttributes` and `PluginParticipantSecurityAttributes` returned by the `AccessControl::get_participant_sec_attributes` operation on the `DomainParticipant`. Specifically:

- The attribute *is_rtps_protected* attribute in the `ParticipantSecurityAttributes` shall be set to FALSE if the value specified in the **<rtps_protection_kind>** element is NONE and to TRUE otherwise.
- The attribute *is_rtps_encrypted* in the `PluginParticipantSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **< rtps_protection_kind>** is ENCRYPT or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.

- The attribute *is_rtps_origin_authenticated* in the `PluginParticipantSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **<rtps_protection_kind>** is SIGN_WITH_ORIGIN_AUTHENTICATION or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.

9.4.1.2.5.7 Topic Access Rules Section

This element is delimited by the XML element **<topic_access_rules>** and contains a sequence of topic rule elements.

9.4.1.2.6 Topic Rule Section

This element is delimited by the XML element **<topic_rule>** and appears within the domain rule Section.

Each topic rule Section contains the following elements:

1. Topic expression

2. Enable Discovery protection

3. Enable Liveliness protection

4. Enable Read Access Control element

5. Enable Write Access Control element

6. Metadata protection Kind

7. Data protection Kind

The contents and delimiters of each Section are described below.

The topic expression element within the rules selects a set of Topic names. The rule applies to any DataReader or DataWriter associated with a Topic whose name matches the Topic expression name.

The topic access rules shall be evaluated in the same order as they appear within the **<topic_access_rules>** Section. If multiple rules match the first rule that matches is the only one that applies.

9.4.1.2.6.1 Topic expression element

This element is delimited by the XML element **<topic_expression>**.

The `value` in this element identifies the set of DDS `Topic` names to which the rule applies. The rule will apply to any `DataReader` or `DataWriter` associated with a `Topic` whose name matches the value.

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 [38].

9.4.1.2.6.2 Enable Discovery protection element

This element is delimited by the XML element **<enable_discovery_protection>**.

This element may take the boolean values TRUE or FALSE.

The setting controls the contents of the `TopicSecurityAttributes` returned by the `AccessControl::get_topic_sec_attributes` on a `Topic` whose associated `Topic` name matches the rule's topic expression. Specifically the **is_discovery_protected** attribute in the `TopicSecurityAttributes` shall be set to the boolean value specified in the **<enable_discovery_protection>** element.

9.4.1.2.6.3 Enable Liveliness Protection element

This element is delimited by the XML element **<enable_liveliness_protection>**.

This element may take the boolean values TRUE or FALSE.

The setting controls the contents of the `TopicSecurityAttributes` returned by the `AccessControl::get_topic_sec_attributes` operation on a `Topic` whose associated `Topic` name matches the rule's topic expression. Specifically the **is_liveliness_protected** attribute in the `TopicSecurityAttributes` shall be set to the boolean value specified in the **<enable_liveliness_protection>** element.

9.4.1.2.6.4 Enable Read Access Control element

This element is delimited by the XML element **<enable_read_access_control>**.

This element may take the boolean values TRUE or FALSE.

The setting shall control the contents of the `TopicSecurityAttributes` returned by the `AccessControl::get_topic_sec_attributes` operation on any `Topic` whose associated `Topic` name matches the rule's topic expression. Specifically the **is_read_protected** attribute in the `TopicSecurityAttributes` shall be set to the boolean value specified in the **<enable_read_access_control>** element.

In addition, this element shall control the `AccessControl::check_create_datareader` operation on any `DataReader` entity whose associated `Topic` name matches the rule's topic expression. Specifically:

- If the value of **<enable_write_access_control>** element is FALSE, the operation `check_create_datareader` shall return TRUE without further checking the Permissions document.
- If the value of **<enable_write_access_control>** element is TRUE, the operation `check_create_datareader` shall return a value according to what is specified in the Permissions document, see 9.4.1.3.

9.4.1.2.6.5 Enable Write Access Control element

This element is delimited by the XML element **<enable_write_access_control>**.

This element may take the boolean values TRUE or FALSE.

The setting shall control the contents of the `TopicSecurityAttributes` returned by the `AccessControl::get_topic_sec_attributes` operation on any `Topic` whose associated `Topic` name matches the rule's topic expression. Specifically the **is_write_protected** attribute in the `TopicSecurityAttributes` shall be set to the binary value specified in the **<enable_write_access_control>** element.

In addition, this element shall control the `AccessControl::check_create_datawriter` operation on any `DataWriter` entity whose associated `Topic` name matches the rule's topic expression. Specifically:

- If the value of **<enable_write_access_control>** element is FALSE, the operation `check_create_datawriter` shall return TRUE without further checking the Permissions document.
- If the value of **<enable_write_access_control>** element is TRUE, the operation `check_create_datawriter` shall return a value according to what is specified in the Permissions document, see 9.4.1.3.

9.4.1.2.6.6 Metadata Protection Kind element

This element is delimited by the XML element **<metadata_protection_kind>**.

This element may take the Protection Kind values NONE, SIGN, ENCRYPT, SIGN_WITH_ORIGIN_AUTHENTICATION, or ENCRYPT_WITH_ORIGIN_AUTHENTICATION.

The setting of this element shall specify the protection kind (see 9.4.1.2.2) used for the RTPS `SubMessages` sent by any `DataWriter` and `DataReader` whose associated `Topic` name matches the rule's topic expression.

The setting of this element shall also control the contents of the `EndpointSecurityAttributes` and `PluginEndpointSecurityAttributes` returned by the `AccessControl::get_datawriter_sec_attributes` and `AccessControl::get_datareader_sec_attributes` operation on any `DataWriter` or `DataReader` entity whose associated `Topic` name matches the rule's topic expression. Specifically:

- The attribute *is_submessage_protected* in the `EndpointSecurityAttributes` shall be set to FALSE if the value specified in the **<metadata_protection_kind>** is NONE and shall be set to TRUE otherwise.
- The attribute *is_submessage_encrypted* in the `PluginEndpointSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **<metadata_protection_kind>** is ENCRYPT or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.
- The attribute *is_submessage_origin_authenticated* in the `PluginEndpointSecurityAttributes` (see 9.4.2.5) shall be set to TRUE if the value specified in the **<metadata_protection_kind>** is SIGN_WITH_ORIGIN_AUTHENTICATION or ENCRYPT_WITH_ORIGIN_AUTHENTICATION and to FALSE otherwise.

9.4.1.2.6.7 Data Protection Kind element

This element is delimited by the XML element **<data_protection_kind>**.

This element may take the Basic Protection Kind values: NONE, SIGN, or ENCRYPT.

The setting of this element shall specify the basic protection kind (see 9.4.1.2.1) used for the RTPS `SerializedPayload` submessage element sent by any `DataWriter` whose associated `Topic` name matches the rule's topic expression.

The setting shall control the contents of the `EndpointSecurityAttributes` and `PluginEndpointSecurityAttributes` returned by the `AccessControl::get_datawriter_sec_attributes` operation on any `DataWriter`

entity whose associated `Topic` name matches the rule's topic expression. Specifically the `PluginEndpointSecurityAttributes` attributes *is_payload_protected* and *is_key_protected*, as well as the `PluginEndpointSecurityAttributes` attribute *is_payload_encrypted* (see 0):

- If the value specified in the **<data_protection_kind>** element is NONE, then *is_payload_protected, is_key_protected* and *is_payload_encrypted* shall be set to FALSE.
- If the value specified in the **<data_protection_kind>** element is SIGN, then *is_payload_protected* shall be set to TRUE. The attributes *is_key_protected* and *is_payload_encrypted* shall be set to FALSE.
- If the value specified in the **<data_protection_kind>** element is ENCRYPT, then *is_payload_protected*, *is_key_protected*, and *is_payload_encrypted* shall be set to TRUE.

9.4.1.2.7 Application of Domain and Topic Rules

For a given `DomainParticipant` the Domain Rules shall be evaluated in the same order they appear in the Governance document. The first Domain Rule having a **<domains>** element whose value matches the `DomainParticipant` domain_id shall be the one applied to the `DomainParticipant`.

If no Domain Rule matches the `DomainParticipant` domain_id the operation under consideration shall fail with a suitable "permissions error". If desired, to avoid this situation, a "default" Domain Rule can be added to the end using the expression:

```
<domains>
    <id_range>
        <min>0</min>
    </id_range>
</domains>
```

This rule will match any domain_id not matched by the rules that appear before.

For a given `Topic`, `DataWriter` or `DataReader` DDS `Entity` belonging to a `DomainParticipant` the Topic Rules appearing within the Domain Rule that applies to that `DomainParticipant` shall be evaluated in the same order they appear in the Governance document. The first Topic Rule having a **<topic_expression>** element whose value matches the topic name associated with the `Entity` shall be the one applied to the `Entity`.

If no Topic Rule matches the `Entity` topic name the operation under consideration shall fail with a suitable "permissions error". If desired, to avoid this situation, a "default" Topic Rule can be added to the end using the expression **<topic_expression>*</ topic_expression >**. This rule will match any topic name not matched by the rules that appear before.

9.4.1.2.8 Example Domain Governance document (non normative)

Following is an example permissions document that is written according to the XSD described in the previous sections.

```
<?xml version="1.0" encoding="utf-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.omg.org/spec/DDS-
Security/20170801/omg_shared_ca_domain_governance.xsd">
    <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>
    <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>NONE
            </metadata_protection_kind>
            <data_protection_kind>NONE
            </data_protection_kind>
        </topic_rule>
```

```
                <topic_rule>
                    <topic_expression>*</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_access_rules>
        </domain_rule>
    </domain_access_rules>
</dds>
```

9.4.1.3 DomainParticipant permissions document

The permissions document is an XML document containing the permissions of the domain participant and binding them to the distinguished name of the `DomainParticipant` as defined in the DDS:Auth:PKI-DH authentication plugin.

The permissions document shall be signed by the Permissions CA. The signed document shall use S/MIME version 3.2 format as defined in IETF RFC 5761 using SignedData Content Type (section 2.4.2 of IETF RFC 5761) formatted as multipart/signed (section 3.4.3 of IETF RFC 5761). This corresponds to the mime-type application/pkcs7-signature. Additionally the signer certificate shall be included within the signature.

The signed permissions document shall be provided to the plugins using the `PropertyQosPolicy` on the `DomainParticipantQos` as specified in Table 56.

The format of this document is defined using the following XSD.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xs:element name="dds" type="PermissionsNode" />
    <xs:complexType name="PermissionsNode">
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="permissions" type="Permissions" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="Permissions">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="grant" type="Grant" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Grant">
```

```xml
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="subject_name" type="xs:string" />
            <xs:element name="validity" type="Validity" />
            <xs:sequence minOccurs="1" maxOccurs="unbounded">
              <xs:choice minOccurs="1" maxOccurs="1">
                <xs:element name="allow_rule" minOccurs="0" type="Rule" />
                <xs:element name="deny_rule" minOccurs="0" type="Rule" />
              </xs:choice>
            </xs:sequence>
            <xs:element name="default" type="DefaultAction" />
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>

    <xs:complexType name="Validity">
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="not_before" type="xs:dateTime" />
            <xs:element name="not_after" type="xs:dateTime" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="Rule">
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:element name="domains" type="DomainIdSet" />
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element name="publish" type="Criteria" />
            </xs:sequence>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element name="subscribe" type="Criteria" />
            </xs:sequence>
             <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element name="relay" type="Criteria" />
            </xs:sequence>
        </xs:sequence>
    </xs:complexType>

<xs:complexType name="DomainIdSet">
        <xs:choice minOccurs="1" maxOccurs="unbounded">
            <xs:element name="id" type="DomainId" />
            <xs:element name="id_range" type="DomainIdRange" />
        </xs:choice>
    </xs:complexType>

<xs:simpleType name="DomainId">
        <xs:restriction base="xs:nonNegativeInteger" />
    </xs:simpleType>

<xs:complexType name="DomainIdRange">
        <xs:choice>
            <xs:sequence>
                <xs:element name="min" type="DomainId" />
                <xs:element name="max" type="DomainId" minOccurs="0" />
            </xs:sequence>
            <xs:element name="max" type="DomainId" />
```

```xml
            </xs:choice>
        </xs:complexType>

        <xs:complexType name="Criteria">
            <xs:all minOccurs="1">
                <!--  DDSSEC11-56  -->
                <xs:element name="topics" minOccurs="1"
                            type="TopicExpressionList" />
                <xs:element name="partitions" minOccurs="0"
                            type="PartitionExpressionList" />
                <xs:element name="data_tags" minOccurs="0" type="DataTags" />
            </xs:all>
        </xs:complexType>

<xs:complexType name="TopicExpressionList">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="topic" type="TopicExpression" />
        </xs:sequence>
    </xs:complexType>

<xs:complexType name="PartitionExpressionList">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="partition" type="PartitionExpression" />
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="TopicExpression">
        <xs:restriction base="xs:string" />
    </xs:simpleType>

    <xs:simpleType name="PartitionExpression">
        <xs:restriction base="xs:string" />
    </xs:simpleType>

    <xs:complexType name="DataTags">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="tag" type="TagNameValuePair" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TagNameValuePair">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="name" type="xs:string" />
            <xs:element name="value" type="xs:string" />
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="DefaultAction">
        <xs:restriction base="xs:string">
            <xs:enumeration value="ALLOW" />
            <xs:enumeration value="DENY" />
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

9.4.1.3.1 Permissions Section

The XML permissions document contains a permissions Section. This is the portion of the XML document delimited by the `<permissions>` XML element tag.

The permissions Section contains a set of grant sections.

9.4.1.3.2 Grant Section

The grant sections appear within the permissions Section delimited by the **<grant>** XML element tag.

Each grant Section contains three sections:

1. Subject name Section (subject_name element)

2. Validity Section (validity element)

3. Rules Section (allow, deny and default elements)

The contents and delimiters of each Section are described below.

9.4.1.3.2.1 Subject name Section

This Section is delimited by the XML element **<subject_name>**.

The subject name Section 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 shall be the x.509 subject name for the `DomainParticipant` as is given in its Authorization Certificate. A permissions Section with a subject name that does not match the subject name given in the corresponding Authorization certificate shall be ignored.

The X.509 subject name is a set of name-value pairs. The format of x.509 subject name shall be the string representation of the X.509 certificate Subject name as defined in IETF RFC 4514 "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names" [51].

For example:
```
<subject_name>emailAddress=cto@acme.com, CN=DDS Shapes Demo, OU=CTO Office,
O=ACME Inc., L=Sunnyvale, ST=CA, C=US</subject_name>
```

9.4.1.3.2.2 Validity Section

This Section is delimited by the XML element **<validity>**. The contents of this element reflect the valid dates for the permissions. It contains both the starting date and the end date using the format defined by `dateTime` data type as specified in sub clause 3.3.7 of [XSD]. Time zones that aren't specified are considered UTC.

A permissions Section with a validity date that falls outside the current date at which the permissions are being evaluated shall be ignored.

9.4.1.3.2.3 Rules Section

This Section contains the permissions assigned to the `DomainParticipant`. It is described as a set of rules.

The rules are applied in the same order that appear in the document. If the criteria for the rule matches the `domain_id` join and/or publish or subscribe operation that is being attempted, then the allow or deny decision is applied. If the criteria for a rule does not match the operation being attempted, the evaluation shall proceed to the next rule. If all rules have been examined without a match, then the decision specified by the "default" rule is applied. The default rule, if present, must appear after all allow and deny rules. If the default rule is not present, the implied default decision is DENY.

The matching criteria for each rule specify the `domain_id`, topics (published and subscribed), the partitions (published and subscribed), and the data-tags associated with the `DataWriter` and `DataReader`.

For the grant to match there shall be a match of the topics, partitions, and data-tags criteria. This is interpreted as an AND of each of the criteria. For a specific criterion to match (e.g., **<topics>**) it is enough that one of the topic expressions listed matches (i.e., an OR of the expressions with the **<topics>** section).

9.4.1.3.2.3.1   Format of the allow rules

Allow rules appear inside the **<allow_rule>** XML Element. Each rule contains a Domains Section; (9.4.1.3.2.3.1.1), followed by a set of allowed actions. There are three kinds of allowed actions: publish, subscribe and relay.

9.4.1.3.2.3.1.1   Domains Section

This Section is delimited by the XML element **<domains>**.

The value in this element identifies the collection of DDS domain_id values to which the rule applies. The syntax is the same as for the domain section of the Governance document. See subclause 9.4.1.2.5.1.

For example:
```
<domains>
    <id>0</id>
</domains>
```

9.4.1.3.2.3.1.2   Format of the Allowed Actions sections

The sections for each of the three action kinds have similar format. The only difference is the name of the XML element used to delimit the action:

- The **Allow Publish** Action is delimited by the **<publish>** XML element

- The **Allow Subscribe** Action is delimited by the **<subscribe>** XML element

- The **Allow Relay** Action is delimited by the **<relay>** XML element

Each allowed action logically contains three orthogonal conditions. These cover the topic name, partitions, and data-tags. All these conditions must be met for the allowed action to apply. Note that some of these conditions may not appear explicitly in the XML file. In this case a specified default value is assumed and applied as if the condition had been explicitly listed.

Each of these three conditions appears in a separate section:

- Allowed Topics Condition section

- Allowed Partitions Condition section

- Allowed Data Tags Condition section

Example:
```
<publish>          <!— delimits the publish action  -->
    <topics>       <!— delimits the topic condition -->
        <topic>Square</topic>
    </topics>
    <partitions>  <!— delimits the partition condition -->
        <partition>A_partition</partition>
    </partitions>
                   <!— data tags condition absent so use default -->
</publish>
```

9.4.1.3.2.3.1.3   Allowed Topic condition section

The *topic condition section* is delimited by the **<topics>** XML element. It defines the DDS Topic names that must be matched for the allow 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 [38].

In order for an action (e.g. a publish action) to be allowed it must meet the *topic condition.* For this to happen the Topic name associated with the intended action must match one the topics or topic expressions explicitly listed in the *topic condition section*.

The *topic condition section* must always be present; therefore there is no default specfied.

Example (appearing within a <allow_rule> and within a publish, subscribe, or relay 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".

9.4.1.3.2.3.1.4   Allowed Partitions condition section

The *allowed partitions condition section* is delimited by the **<partitions>** XML element. It limits the set DDS Partitions names that may be associated with the (publish, subscribe, relay) action for the rule to apply. Partition names may be given explicitly or by means of Partition name expressions. Each partition name or partition-name expression appears separately in a **<partition>** sub-element within the **<partitions>** element.

The Partition 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 [38].

In order for an action (e.g. a publish action) to meet the *allowed partitions condition* that appears within an allow rule, the set of the Partitions associated with the DDS entity (DataWriter or

DataReader) attempting the (publish, subscribe, or relay) action must be contained in the set of partitions defined by the ***allowed partitions condition section***.

If there is no **<partitions>** Section within an allow rule, then the default "empty string" partition is assumed. See PARTITION QosPolicy entry in Qos Policies table of section 2.2.3 (Supported Qos) of the DDS Specification version 1.4. This means that the allow rule (e.g. publish) would only allow a DataWriter to publish 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 attempting to perform the action (e.g. publish action) is 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}.

For legacy reasons DDS-Security implementations shall provide a way to select an alternative "legacy matching" behavior. The "legacy matching behavior" shall match the ***allowed partitions condition*** condition as long as one or more of the Partitions associated with DDS Entity attempting to perform the action (e.g. DataWriter for a publish action) matches one of the partitions in the ***allowed partitions condition***. The same ***allowed partitions condition section*** above would be matched if the partitions associated with the DDS DataWriter include A or B. So it would be OK to publish in A, in B, or in {A, B} and also in {A, B, C}.

9.4.1.3.2.3.1.5   Allowed Data tags condition section

The ***allowed data tags condition section*** is delimited by the **< data_tags>** XML element. It limits the set DDS Data Tags that may be associated with the (publish, subscribe, relay) action for the rule to apply. The **<data_tags>** XML Element contain a set of tags.

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".

```xml
<allow_rule>
        <domains>
            <id>0</id>
        </domains>
        <publish>
            <topics>
                <topic>Cir*</topic>
            </topics>
            <data_tags>
                <tag>
                    <name>aTagName1</name>
                    <value>aTagValue1</value>
                </tag>
            </data_tags>
        </publish>
        <subscribe>
            <topics>
                <topic>Sq*</topic>
            </topics>
            <data_tags>
                <tag>
                    <name>aTagName1</name>
                    <value>aTagValue1</value>
                </tag>
                <tag>
                    <name>aTagName2</name>
                    <value>aTagValue2</value>
                </tag>
            </data_tags>
        </subscribe>
        <subscribe>
            <topics>
                <topic>Triangle</topic>
            </topics>
            <partitions>
                <partition>P*</partition>
            </partitions>
        </subscribe>
</allow_rule>
```

9.4.1.3.2.3.2   Format for deny rules

Deny rules appear inside the **<deny_rule>** XML Element. Each rule contains a Domains Section; (9.4.1.3.2.3.1.1), followed by a set of denied actions. There are three kinds of denied actions: publish, subscribe and relay.

Deny rules have the same format as the allow rules. The only difference is how they are interpreted. If the criteria in the deny rule matches the operation being performed, then the decision is to deny the operation.

This Section is delimited by the XML element **<domains>**. The value in this element identifies the collection of DDS domain_id values to which the rule applies. The syntax is the same as for the domain section of the Governance document. See subclause 9.4.1.2.5.1.

For example:
```
<domains>
    <id>0</id>
</domains>
```

9.4.1.3.2.3.2.2   Format of the Denied Actions sections

The sections for each of the three action kinds have similar format. The only difference is the name of the XML element used to delimit the action:

- The **Deny Publish** Action is delimited by the **<publish>** XML element

- The **Deny Subscribe** Action is delimited by the **<subscribe>** XML element

- The **Deny Relay** Action is delimited by the **<relay>** XML element

Each denied action logically contains three orthogonal deny conditions. These cover the topic name, partitions, and data-tags. All these conditions must be met for the denied action to apply. Note that some of these conditions may not appear explicitly in the XML file. In this case a specified default value is assumed and applied as if the condition had been explicitly listed.

Each of these three conditions appears in a separate section:

- Denied Topics Condition section

- Denied Partitions Condition section

- Denied Data Tags Condition section

Example (appearing within a <deny_rule>):
```
<publish>            <!— delimits the publish action  -->
    <topics>         <!— delimits the topic condition -->
        <topic>Square</topic>
    </topics>
    <partitions>  <!— delimits the partition condition -->
        <partition>A_partition</partition>
    </partitions>
                 <!— data tags condition absent so use default -->
</publish>
```

9.4.1.3.2.3.2.3   Denied Topic condition section

The ***denied topic condition section*** is delimited by the **<topics>** XML element. It has the same format and interpretation as the ***allowed topic condition section*** for the allowed actions, see **Error! Reference source not found.**.

In order for an action (e.g. a publish action) to be denied it must meet the ***denied topic condition.*** For this to happen the Topic name associated with the intended action must match one the topics or topic expressions explicitly listed in the ***denied topic condition section***.

The ***denied partitions condition section*** is delimited by the **\<partitions>** XML element. It defines the DDS Partitions names that when associated with the (publish, subscribe, relay) cause the deny action for the rule to apply. Partition names may be given explicitly or by means of Partition name expressions. Each partition name or partition-name expression appears separately in a **\<partition>** sub-element within the **\<partitions>** element.

In order for an action (e.g. a publish action) to be denied it must meet the ***denied partitions condition.*** For this to happen one of more of the partition names associated with the DDS Entity performing the action (e.g. a DataWriter for the publish action) must match one the partitions or partition expressions explicitly listed in the ***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 independent of the partition associated with the DDS Endpoint (DataWriter for the 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 C, but not in {A}, {A, B}, or {A, B, C}.

9.4.1.3.2.3.2.5   Data tags condition section

The ***denied data tags condition section*** is delimited by the **\<data_tags>** XML element. It defines the DDS tags names and values that when associated with the (publish, subscribe, relay) cause the deny action for the rule to apply.

In order for an action (e.g. a publish action) to be denied it must meet the ***denied data tags condition.*** For this to happen the DDS Entity associated with the action (e.g. DataWriter for a publish action) must have a data tag name and value pair associated that matches one 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 independent 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 with two associated data-tags { (aTagName1, aTagValue1), (aTagName2, aTagValue2)}.

9.4.1.3.2.3.2.6   Example deny rule

```
<allow_rule>
        <domains>
            <id>0</id>
        </domains>
        <publish>
            <topics>
                <topic>Cir*</topic>
            </topics>
            <data_tags>
                <tag>
                    <name>aTagName1</name>
                    <value>aTagValue1</value>
                </tag>
            </data_tags>
        </publish>
        <subscribe>
            <topics>
                <topic>Sq*</topic>
            </topics>
            <data_tags>
                <tag>
                    <name>aTagName1</name>
                    <value>aTagValue1</value>
                </tag>
                <tag>
                    <name>aTagName2</name>
                    <value>aTagValue2</value>
                </tag>
            </data_tags>
        </subscribe>
        <subscribe>
            <topics>
                <topic>Triangle</topic>
            </topics>
            <partitions>
                <partition>P*</partition>
            </partitions>
        </subscribe>
    </allow_rule>
```

9.4.1.3.2.3.2.7   Example deny rule

```
<deny_rule>
        <domains>
            <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>
                <data_tags>
                    <tag>
                        <name>aTagName1</name>
                        <value>aTagValue1</value>
                    </tag>
                    <tag>
                        <name>aTagName2</name>
                        <value>aTagValue2</value>
                    </tag>
                </data_tags>
            </subscribe>
    </deny_rule>
```

9.4.1.4   DomainParticipant example permissions document (non normative)

Following is an example permissions document that is written according to the XSD described in the previous sections.

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.omg.org/spec/DDS-
Security/20170801/omg_shared_ca_permissions.xsd">
  <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[Z|(+|-)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>2013-10-26T00:00:00</not_before>
        <not_after>2018-10-26T22:45:30</not_after>
```

```xml
    </validity>
    <allow_rule>
      <domains>
        <id>0</id>
      </domains>
      <!--  DDSSEC11-56 - deleted invalid elements -->
    </allow_rule>
    <deny_rule>
      <domains>
        <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>
    <allow_rule>
      <domains>
        <id>0</id>
      </domains>
      <publish>
        <topics>
          <topic>Cir*</topic>
        </topics>
        <data_tags>
          <tag>
            <name>aTagName1</name>
            <value>aTagValue1</value>
          </tag>
        </data_tags>
      </publish>
      <subscribe>
```

```
            <topics>
                <topic>Sq*</topic>
            </topics>
            <data_tags>
                <tag>
                    <name>aTagName1</name>
                    <value>aTagValue1</value>
                </tag>
                <tag>
                    <name>aTagName2</name>
                    <value>aTagValue2</value>
                </tag>
            </data_tags>
        </subscribe>
        <subscribe>
            <topics>
                <topic>Triangle</topic>
            </topics>
            <partitions>
                <partition>P*</partition>
            </partitions>
            <data_tags>
                <tag>
                    <name>aTagName1</name>
                    <value>aTagValue1</value>
                </tag>
            </data_tags>
        </subscribe>
        <relay>
            <topics>
                <topic>*</topic>
            </topics>
            <partitions>
                <partition>aPartitionName</partition>
            </partitions>
        </relay>
    </allow_rule>
    <default>DENY</default>
    </grant>
  </permissions>
</dds>
```

9.4.2 DDS:Access:Permissions Types

This sub clause specifies the content and format of the `Credential` and `Token` objects used by the DDS:Access:Permissions plugin.

9.4.2.1   DDS:Access:Permissions PermissionsCredentialToken

The DDS:Access:Permissions plugin shall set the attributes of the `PermissionsCredentialToken` object as specified in the table below.

Table 57 – PermissionsCredentialToken class for the builtin AccessControl plugin

Attribute name	Attribute value	
class_id	"DDS:Access:PermissionsCredential"	
properties	*name*	*value*
	dds.perm.cert	Contents of the permissions document signed by the PermissionCA that was configured using the Participant PropertyQosPolicy with name "dds.sec.access.permissions"

9.4.2.2 DDS:Access:Permissions PermissionsToken

The DDS:Access:Permissions plugin shall set the attributes of the `PermissionsToken` object as specified in the table below:

Table 58 – PermissionsToken class for the builtin AccessControl plugin

Attribute name	Attribute value	
class_id	"DDS:Access:Permissions:1.0"	
properties	*name*	*value*
(The presence of each of these properties is optional)	dds.perm_ca.sn	The subject name of Permissions CA
	dds.perm_ca.algo	"RSA-2048" or "EC-prime256v1"

The value of the *class_id* shall be interpreted as composed of three parts: a *PluginClassName*, a *MajorVersion* and a *MinorVersion* according to the same format described in clause 9.3.2.1.

Accordingly this version of the specification has *PluginClassName* equal to "DDS:Access:Permissions", *MajorVersion* set to 1, and *MinorVersion* set to 0.

If the *MajorVersion* and *MinorVersion* are missing from the class_id, it shall be interpreted as being *MajorVersion* 1 and *MinorVersion* 0.

9.4.2.3 PluginParticipantSecurityAttributes

The `PluginParticipantSecurityAttributes` describe plugin-specific behavior of the builtin DDS:Crypto:AES:GCM-GMAC Crypto affecting the key material and transformations for the RTPS messages and the RTPS submessages related to the builtin Topics.

This is a structured type, whose members are described in the table below:

Table 59 – Description of the PluginParticipantSecurityAttributes

Member	Type	Meaning
is_rtps_encrypted	Boolean	This field is only used if the ParticipantSecurityAttributes field *is_rtps_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.
		This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the RTPS message shall be protected using authenticated encryption or only an authentication code.
		If *is_rtps_encrypted* is TRUE, the CryptoKeyFactory `register_local_participant` operation shall create key material for performing a GCM authenticated encryption and the CryptoTransform `encode_rtps_message` operation shall apply the GCM authenticated encryption transformation.
		If *is_rtps_encrypted* is FALSE operation the aforementioned operations shall create key material for performing a GMAC authentication and the CryptoTransform `encode_rtps_message` operation shall apply the GMAC authentication transformation.
is_discovery_encrypted	Boolean	This field is only used if the ParticipantSecurityAttributes field *is_discovery_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.
		This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the submessages related to the **builtin secure discovery endpoints** (see 7.4.6) shall be protected using authenticated encryption or only an authentication code.
		If *is_discovery_encrypted* is TRUE, the CryptoKeyFactory `register_local_datawriter` (in the case of a DataWriter endpoint) or `register_local_datareader` (in the case of a DataReader endpoint) operation for the **builtin secure discovery endpoints** shall create key material for performing a GCM authenticated encryption and the CryptoTransform `encode_datawriter_submessage` and `encode_datareader_submessage` operations shall apply the GCM authenticated encryption transformation.
		If *is_discovery_encrypted* is FALSE operation the aforementioned operations shall create key material for performing a GMAC authentication and the CryptoTransform `encode_rtps_submessage` and `encode_datawriter_submessage` operations shall apply the GMAC authentication transformation.

is_liveliness_encrypted	Boolean	This field is only used if the ParticipantSecurityAttributes field *is_liveliness_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.
		This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the submessages related to the **builtin secure liveliness endpoints** (see 7.4.7) shall be protected using authenticated encryption or only an authentication code.
		If *is_liveliness_encrypted* is TRUE, the CryptoKeyFactory `register_local_datawriter` (in the case of a DataWriter endpoint) or `register_local_datareader` (in the case of a DataReader endpoint) operation for the **builtin secure liveliness endpoints** shall create key material for performing a GCM authenticated encryption and the CryptoTransform `encode_datawriter_submessage` and `encode_datareader_submessage` operations shall apply the GCM authenticated encryption transformation.
		If *is_liveliness_encrypted* is FALSE operation the aforementioned operations shall create key material for performing a GMAC authentication and the CryptoTransform `encode_datawriter_submessage` and `encode_datareader_submessage` operations shall apply the GMAC authentication transformation.
is_rtps_origin_authenticated	Boolean	This field is only used if the ParticipantSecurityAttributes field *is_rtps_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.
		This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the RTPS messages shall have additional authentication codes constructed using receiver-specific keys.
		If *is_rtps_origin_authenticated* is TRUE, the CryptoKeyFactory `register_matched_remote_participant` operation shall create additional receiver-specific key material for performing a GMAC authenticatication. The CryptoTransform `encode_rtps_message` operation shall add additional GMAC authentication codes using the receiver-specific key material.
		If *is_rtps_origin_authenticated* is FALSE, the aforementioned operations shall not create additional key material and the CryptoTransform `encode_rtps_message` shall not add additional GMAC authentication codes.

| is_discovery_origin_au thenticated | Boolean | This field is only used if the ParticipantSecurityAttributes field *is_discovery_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.

This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the RTPS submessage from or to the **builtin secure discovery endpoints** shall have additional authentication codes constructed using receiver-specific keys.

If *is_discovery_origin_authenticated* is TRUE, the CryptoKeyFactory `register_matched_datareader` (in the case of a DataWriter endpoint) or `register_matched_datawriter` (in the case of a DataReader endpoint) operation shall create additional receiver-specific key material for performing a GMAC authentication. The CryptoTransform `encode_datawriter_submessage` and `encode_datareader_submessage` operations shall add additional GMAC authentication codes using the receiver-specific key material.

If *is_discovery_origin_authenticated* is FALSE, the aforementioned operations shall not create additional key material and the CryptoTransform `encode_datawriter_submessage` and `encode_datareader_submessage` operations shall not add additional GMAC authentication codes. |
| is_liveliness_origin_au thenticated | Boolean | This field is only used if the ParticipantSecurityAttributes field *is_liveliness_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.

This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the RTPS submessage from or to the **builtin secure liveliness endpoints** shall have additional authentication codes constructed using receiver-specific keys.

If *is_liveliness_origin_authenticated* is TRUE, the CryptoKeyFactory `register_matched_datareader` (in the case of a DataWriter endpoint) or `register_matched_datawriter` (in the case of a DataReader endpoint) operation shall create additional receiver-specific key material for performing a GMAC authenticatication. The CryptoTransform `encode_datawriter_submessage` and `encode_datareader_submessage` operations shall add additional GMAC authentication codes using the receiver-specific key material. |

		If *is_liveliness_origin_authenticated* is FALSE, the aforementioned operations shall not create additional key material and the CryptoTransform `encode_datawriter_submessage` and `encode_datareader_submessage` operations shall not add additional GMAC authentication codes.

9.4.2.4 Definition of the PluginParticipantSecurityAttributesMask

The `PluginParticipantSecurityAttributesMask` is used to encode the value of the `PluginParticipantSecurityAttributes` in a compact way such that it can be included in the `ParticipantSecurityInfo`, see 7.2.7.

As described in section 7.2.7, in order to communicate, two DomainParticipants need to have the same `ParticipantSecurityInfo`. As as consequence the `PluginParticipantSecurityAttributesMask` must also be the same.

The default value for the mask is:

```
#define   PLUGIN_PARTICIPANT_SECURITY_ATTRIBUTES_MASK_DEFAULT 0
```

The mapping of the `PluginParticipantSecurityAttributes` to `PluginParticipantSecurityAttributesMask` is as follows:

Table 60 – Mapping of PluginParticipantSecurityAttributes to the PluginParticipantSecurityAttributesMask

Field in PluginParticipantSecurityAttributes	Corresponding bit in the `PluginParticipantSecurityAttributesMask`
is_rtps_encrypted	`#define PLUGIN_PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_RTPS_ ENCRYPTED (0x00000001 << 0)`
is_discovery_encrypted	`#define PLUGIN_PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_BUILTIN_ IS_DISCOVERY_ENCRYPTED (0x00000001 << 1)`
is_liveliness_encrypted	`#define PLUGIN_PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_LIVEL INESS_ENCRYPTED (0x00000001 << 2)`
is_rtps_origin_authenticated	`#define PLUGIN_PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_RTPS_ ORIGIN_AUTHENTICATED (0x00000001 << 3)`
is_discovery_origin_authenticated	`#define PLUGIN_PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_DISCO VERY_ORIGIN_AUTHENTICATED (0x00000001 << 4)`

is_liveliness_origin_authenticated	`#define PARTICIPANT_SECURITY_ATTRIBUTES_FLAG_IS_LIVELINESS_ORIGIN_AUTHENTICATED (0x00000001 << 5)`	

9.4.2.5 PluginEndpointSecurityAttributes

The `PluginEndpointSecurityAttributes` describe plugin-specific behavior of the builtin DDS:Crypto:AES:GCM-GMAC Crypto affecting the key material and transformations for endpoints (DataWriters and DataReaders) submessages and submessage payloads.

This is a structured type, whose members are described in the table below:

Table 61 – Description of the PluginEndpointSecurityAttributes

Member	Type	Meaning
is_submessage_encrypted	Boolean	This field is only used if the EndpointSecurityAttributes field *is_submessage_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE. This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the submessage shall be protected using authenticated encryption or only an authentication code. If *is_submessage_encrypted* is TRUE, the CryptoKeyFactory `register_local_datawriter` (in the case of a `DataWriter` endpoint) or `register_local_datareader` (in the case of a `DataReader` endpoint) operation shall create key material for performing a GCM authenticated encryption and the CryptoTransform operation `encode_datawriter_submessage` (in the case of a `DataWriter` endpoint) or `encode_datareader_submessage` (in the case of a `DataReader`) shall apply the GCM authenticated encryption transformation. If *is_submessage_encrypted* is FALSE operation the aforementioned operations shall create key material for performing a GCM authenticated encryption and the CryptoTransform operation `encode_datawriter_submessage` (in the case of a `DataWriter` endpoint) or `encode_datareader_submessage` (in the case of a `DataReader`) shall apply the GCM authenticated encryption transformation.

| is_submessage_origin_aut henticated | Boolean | This field is only used if the EndpointSecurityAttributes field *is_submessage_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.

This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the submessage shall have additional authentication codes constructed using receiver-specific keys.

If *is_submessage_origin_authenticated* is TRUE, the CryptoKeyFactory `register_matched_datareader` (in the case of a DataWriter endpoint) or `register_matched_datawriter` (in the case of a DataReader endpoint) operation shall create additional receiver-specific key material for performing a GMAC authenticatication. The CryptoTransform operation `encode_datawriter_submessage` (in the case of a `DataWriter` endpoint) or `encode_datareader_submessage` (in the case of a `DataReader`) shall add additional GMAC authentication codes using the receiver-specific key material.

If *is_submessage_origin_authenticated* is FALSE, the aforementioned operations shall not create additional key material and the CryptoTransform operation `encode_datawriter_submessage` (in the case of a `DataWriter` endpoint) or `encode_datareader_submessage` (in the case of a `DataReader`) shall not add additional GMAC authentication codes. |
| is_payload_encrypted | Boolean | This field is only used if the EndpointSecurityAttributes field *is_payload_protected* is TRUE. Otherwise it has no effect and it shall be set to FALSE.

This field indicates to the DDS:Crypto:AES:GCM-GMAC plugin whether the payload shall be protected using authenticated encryption or only an authentication code.

If *is_payload_encrypted* is TRUE, the CryptoKeyFactory `register_local_datawriter` (in the case of a DataWriter endpoint) or `register_local_datareader` (in the case of a DataReader endpoint) operation shall create key material for performing a GCM authenticated encryption and the CryptoTransform `encode_serialized_payload` operation shall apply the GCM authenticated encryption transformation. |

		If *is_payload_encrypted* is FALSE operation the aforementioned operations shall create key material for performing a GCM authenticated encryption and the CryptoTransform `encode_serialized_payload` operation shall apply the GCM authenticated encryption transformation.

9.4.2.6 Definition of the PluginEndpointSecurityAttributesMask

The `PluginEndpointSecurityAttributesMask` is used to encode the value of the `PluginEndpointSecurityAttributes` in a compact way such that it can be included in the `EndpointSecurityInfo`, see 7.2.8.

As described in section 7.2.8, in order to communicate, two endpoints need to have the same `EndpointSecurityAttributesMask`. As as consequence the `PluginEndpointSecurityAttributesMask` must also be the same.

The default value for the mask is:

```
#define   PLUGIN_ENDPOINT_SECURITY_ATTRIBUTES_MASK_DEFAULT 0
```

The mapping of the `PluginEndpointSecurityAttributes` to `PluginEndpointSecurityAttributesMask` is as follows:

Table 62 – Mapping of fields PluginEndpointSecurityAttributes to the PluginEndpointSecurityAttributesMask

Field in `PluginEndpointSecurityAttributes`	Corresponding bit in the `PluginEndpointSecurityAttributesMask`
is_submessage_encrypted	#define PLUGIN_ENDPOINT_SECURITY_ATTRIBUTES_FLAG _IS_SUBMESSAGE_ENCRYPTED (0x00000001 << 0)
is_payload_encrypted	#define PLUGIN_ENDPOINT_SECURITY_ATTRIBUTES_FLAG _IS_PAYLOAD_ENCRYPTED (0x00000001 << 1)
is_submessage_origin_authenticated	#define PLUGIN_ENDPOINT_SECURITY_ATTRIBUTES_FLAG _IS_SUBMESSAGE_ORIGIN_AUTHENTICATED (0x00000001 <<2)

9.4.3 DDS:Access:Permissions plugin behavior

The DDS:Access:Permissions shall be initialized to have access to the Permissions CA 2048-bit RSA public key. As this is a builtin plugin the mechanism for initialization is implementation dependent.

The table below describes the actions that the DDS:Access:Permissions plugin performs when each of the plugin operations is invoked.

`check_create_participant`	This operation shall use the ***permissions_handle*** to retrieve the cached Permissions and Governance information.
	If the Governance specifies any topics on the `DomainParticipant` ***domain_id*** with ***enable_read_access_control*** set to FALSE or with ***enable_write_access_control*** set to FALSE, then the operation shall succeed and return TRUE.
	If the `ParticipantSecurityAttributes` has ***is_access_protected*** set to FALSE, then the operation shall succeed and return TRUE.
	Otherwise the operation shall return FALSE.
`check_create_datawriter`	This operation shall use the ***permissions_handle*** to retrieve the cached Permissions and Governance information.
	If the Governance specifies a topic or topic-expression on the `DomainParticipant` ***domain_id*** matching the `DataWriter` topic with ***enable_write_access_control*** set to FALSE, then the operation shall succeed and return TRUE.
	If the Permissions document contains a Grant for the `DomainParticipant` allowing it to publish the `Topic` with specified ***topic_name*** on all the `Publisher`'s `PartitionQosPolicy` names and with all the tags in the `DataWriter DataTagQosPolicy`, then the operation shall succeed and return TRUE.
	Otherwise the operation shall return FALSE.
`check_create_datareader`	This operation shall use the ***permissions_handle*** to retrieve the cached Permissions and Governance information.
	If the Governance specifies a topic or topic-expression on the `DomainParticipant` ***domain_id*** matching the `DataReader` topic with ***enable_read_access_control*** set to FALSE, then the operation shall succeed and return TRUE.
	If the Permissions document contains a Grant for the `DomainParticipant` allowing it to subscribe the `Topic` with specified ***topic_name*** on all the `Subscriber`'s `PartitionQosPolicy` names and with all the tags in the `DataReader DataTagQosPolicy`, then the operation shall succeed and return TRUE.
	Otherwise the operation shall return FALSE.

`check_create_topic`	This operation shall use the ***permissions_handle*** to retrieve the cached Permissions and Governance information. If the Governance specifies a topic or topic-expression on the `DomainParticipant` ***domain_id*** matching the `Topic` name with ***enable_read_access_control*** set to FALSE or with ***enable_write_access_control*** set to FALSE, then the operation shall succeed and return TRUE. If the Permissions document contains a Grant for the `DomainParticipant` allowing it to publish the `Topic` with specified ***topic_name***, then the operation shall succeed and return TRUE. If the Permissions document contains a Grant for the `DomainParticipant` allowing it to subscribe the `Topic` with specified ***topic_name***, then the operation shall succeed and return TRUE. Otherwise the operation shall return FALSE.
`check_local_datawriter_register_instance`	This operation shall return TRUE.
`check_local_datawriter_dispose_instance`	This operation shall return TRUE.
`check_remote_participant`	This operation shall use the ***permissions_handle*** to retrieve the cached local `DomainParticipant` Governance and the remote `DomainParticipant` Permissions information. If the `ParticipantSecurityAttributes` has ***is_access_protected*** set to FALSE, then the operation shall succeed and return TRUE. If the ***PluginClassName*** or the ***MajorVersion*** of the ***local permissions_token*** differ from those in the ***remote_permissions_token***, the operation shall return FALSE. If the Permissions document contains a Grant for the remote `DomainParticipant` and the Grant contains an allow rule on the `DomainParticipant` ***domain_id***, then the operation shall succeed and return TRUE. Otherwise the operation shall return FALSE.

`check_remote_datawriter`	This operation shall use the ***permissions_handle*** to retrieve the cached local `DomainParticipant` Governance and the remote `DomainParticipant` Permissions information.
	If the Governance specifies a topic or topic-expression on the `DomainParticipant` ***domain_id*** matching the remote `DataWriter` topic with ***enable_write_access_control*** set to FALSE, then the operation shall succeed and return TRUE.
	If the ***PluginClassName*** or the ***MajorVersion*** of the ***local permissions_token*** differ from those in the ***remote_permissions_token***, the operation shall return FALSE.
	If the remote `DomainParticipant` Permissions document contains a Grant allowing it to publish the `DataWriter`'s ***topic_name*** on all the remote `Publisher`'s `PartitionQosPolicy` names and with all the tags in the remote `DataWriter` `DataTagQosPolicy`, then the operation shall succeed and return TRUE.
	Otherwise the operation shall return FALSE.
`check_remote_datareader`	This operation shall use the ***permissions_handle*** to retrieve the cached local `DomainParticipant` Governance and the remote `DomainParticipant` Permissions information.
	If the Governance specifies a topic or topic-expression on the `DomainParticipant` ***domain_id*** matching the remote `DataReader` topic with ***enable_read_access_control*** set to FALSE, then the operation shall succeed, set the 'allow_relay_only' output parameter to FALSE, and return TRUE.
	If the ***PluginClassName*** or the ***MajorVersion*** of the ***local permissions_token*** differ from those in the ***remote_permissions_token***, the operation shall return FALSE.
	If the Permissions document contains a Grant for the remote `DomainParticipant` allowing it to subscribe the `DataReader`'s ***topic_name*** on all the `Subscriber`'s `PartitionQosPolicy` names and with all the tags in the `DataReader` `DataTagQosPolicy`, then the operation shall succeed, set the 'allow_relay_only' output parameter to FALSE, and return TRUE.

	If the Permissions document contains a Grant for the remote `DomainParticipant` allowing it to 'relay' the `DataReader`'s *topic_name*, the operation shall return TRUE and also set the 'allow_relay_only' output parameter to TRUE. Otherwise the operation shall return FALSE.
`check_remote_topic`	This operation shall use the ***permissions_handle*** to retrieve the cached local `DomainParticipant` Governance and the remote `DomainParticipant`Permissions information. If the Governance specifies a topic or topic-expression on the `DomainParticipant` *domain_id* matching the `Topic` name with ***enable_read_access_control*** set to FALSE or with ***enable_write_access_control*** set to FALSE, then the operation shall succeed and return TRUE. If the ***PluginClassName*** or the ***MajorVersion*** of the ***local permissions_token*** differ from those in the ***remote_permissions_token***, the operation shall return FALSE. If the Permissions document contains a Grant for the `DomainParticipant` allowing it to publish the `Topic` with specified ***topic_name***, then the operation shall succeed and return TRUE. If the Permissions document contains a Grant for the `DomainParticipant` allowing it to subscribe the `Topic` with specified ***topic_name***, then the operation shall succeed and return TRUE. Otherwise the operation shall return FALSE.
`check_local_datawriter_ma tch`	This operation shall return TRUE.
`check_local_datareader_ma tch`	This operation shall return TRUE.
`check_remote_datawriter_r egister_instance`	This operation shall return TRUE.
`check_remote_datawriter_d ispose_instance`	This operation shall return TRUE.
`get_permissions_token`	This operation shall return the `PermissionsToken` formatted as described in 9.4.2.2.

`get_permissions_credential_token`	This operation shall return the `PermissionsToken` formatted as described in 9.4.2.1
`set_listener`	This operation shall save a reference to the listener object and associate it with the specified `PermissionsHandle`.
`return_permissions_token`	This operation shall behave as specified in 8.4.2.9.20
`return_permissions_credential_token`	This operation shall behave as specified in 8.4.2.9.21
`validate_local_permissions`	This operation shall receive the DomainId and `DomainParticipantQos` from which it can access the Identity Certificate, Signed Domain Governance and Signed Permissions document. The operation shall check the subject name in the Identity Certificate matches the one from the Signed Permissions document. The operation shall verify the signature of the Signed Domain Governance and Signed Permissions document by the configured Permissions CA. If all of these succeed, the operation shall cache the Permissions (see 9.4.1.3.1) from the certificate and return an opaque handle that the plugin can use to refer to the saved information. Otherwise the operation shall return an error.
`validate_remote_permissions`	This operation shall invoke the operation `get_authenticated_peer_credential_token` on the ***auth_plugin*** passing the ***remote_identity_handle*** to retrieve the `AuthenticatedPeerCredentialToken` (see 0) for the remote `DomainParticipant`. The `AuthenticatedPeerCredentialToken` contains both the Identity Certificate and the Signed Permissions Document obtained from the remote DomainParticipant during the Authentication. The operation shall check the subject name in the Signed Permissions Document matches the one in the Identity Certificate. The operation shall verify the signature of the Signed Permissions Document by the configured Permissions CA. If all of these succeed, the operation shall cache the Permission Section from the Signed Permissions Document and return an opaque handle that the plugin can use to refer to the saved information. Otherwise the operation shall return an error.

get_participant_sec_attri butes	This operation shall use the ***permissions_handle*** to retrieve the cached Permissions and Governance information.
	Based on the Governance document rules for the `DomainParticipant` ***domain_id*** the operation shall fill the ***attributes*** output parameter. The fields of the `ParticipantSecurityAttributes` ***attributes*** shall be set according to the following rules:
	If the Governance document has the element ***allow_unauthenticated_participants*** set to FALSE, the ***attributes*** field ***allow_unauthenticated_participants*** shall be set to FALSE. Otherwise the field shall be set to TRUE.
	If the Governance document has the element ***enable_join_access_control*** set to FALSE, the ***attributes*** field ***is_access_protected*** shall be set to FALSE. Otherwise the field shall be set to TRUE.
	If the Governance document has the element ***rtps_protection_kind*** set to NONE, the ***attributes*** field ***is_rtps_protected*** shall be set to FALSE. Otherwise the field shall be set to TRUE.
return_participant_sec_at tributes	This operation shall behave as specified in 8.4.2.9.26.
return_datawriter_sec_att ributes	This operation shall behave as specified in 8.4.2.9.27.
.return_datareader_sec_at tributes	This operation shall behave as specified in 8.4.2.9.28.

9.5   Builtin Crypto: DDS:Crypto:AES-GCM-GMAC

The builtin `Cryptographic` plugin is referred to as "DDS:Crypto:AES-GCM-GMAC" plugin.

DDS:Crypto:AES-GCM-GMAC provides authenticated encryption using Advanced Encryption Standard (AES) in Galois Counter Mode (AES-GCM) [45]. It supports two AES key sizes: 128 bits and 256 bits. It may also provide additional reader-specific message authentication codes (MACs) using Galois MAC (AES-GMAC) [45].

The definition of the AES-GCM and AES-GMAC transformations shall be as specified in NIST SP 800-38D [45] specialized to 128-bit and 256-bit AES keys with 96-bit Initialization Vector. The most relevant aspects are summarized below.

The AES-GCM authenticated encryption operation is a transformation that takes the four inputs and produces two outputs, symbolically:

```
C, T =  AES-GCM(K, P, AAD, IV)
```

The AES-GCM inputs are described in Table 64 below.

Table 64 – AES-GCM transformation inputs

Input	Description
K	The 128-bit key to be used with the AES-128 block cipher or the 256-bit key to be used with the AES-256 block cipher.
P	The plaintext. This is the data to encrypt and authenticate. It may be empty in case we only want to authenticate data.
AAD	Additional Authenticated Data. This is data beyond the plaintext that will only be authenticated. I.e. it is not encrypted.
IV	Initialization Vector. This is a 96-bit NONCE that shall not be repeated for the same key.

The AES-GCM transformation outputs are described in Table 65 below.

Table 65 – AES-GCM trasnsformation outputs

Input	Description
C	Ciphertext. This is the encryption of the plaintext "P".
T	Authentication Tag. This is a Message Authentication Code (MAC) that provides authentication for the Ciphertext (C) and the Additional Authenticated Data (AAD).

AES-GCM uses AES in counter mode with a specific incrementing function called "inc32" used to generate the counter blocks. As recommended in section 5.2.1.1 of NIST SP 800-38D [45] the counter blocks shall be created from the 96-bit Initialization Vector as follows:

- The initial value of the 128-bit counter block is a 128-bit string containing the IV as the leading 96 bits and zeros the remaining right-most 32 bits.
- Incremental values of the 128-bit counter block used to encrypt each block are obtained using the "inc32" function which increments the right-most 32 bits of the string, regarded as the binary representation of a big-endian integer, modulo 2^32. The inc32 operation does not touch the leading 96 bits.

The AES-GMAC transformation is defined as the special case where the plaintext "P" is empty (zero length). This transformation produces only an AuthenticationTag (Message Authentication Code) on the AAD data:

```
T =  AES-GMAC(K, AAD, IV) = AES-GCM(K, "", AAD, IV)
```

The use of (Galois) counter mode allows authenticated decryption of blocks in arbitrary order. All that is needed to decrypt and validate the authentication tag are the Key and the Initialization Vector. This is very important for DDS because a `DataReader` may not receive all the samples written by a matched `DataWriter`. The use of DDS `ContentFilteredTopics` as well as DDS QoS policies such as `History` (with KEEP_LAST kind), `Reliability` (with BEST_EFFORTS kind), `Lifespan`, and `TimeBasedFilter`, among others, can result in a `DataReader` receiving a subset of the samples written by a `DataWriter`.

The AES-GCM transformation produces both the ciphertext and a message authentication code (MAC) using the same secret key. This is sufficient to protext the plaintext and ensure integrity. However there are situations where multiple MACs are required. For example when a DataWriter shares the same Key with multiple DataReaders and, in spite of this, the DataWriter needs to ensure message-origin authentication. In this situation the DataWriter should create a separate "reader-specific key" used only for authentication and append additional reader-specific MACs, each computed with one of the reader-specific keys.

9.5.1 Configuration

The DDS:Crypto:AES-GCM-GMAC plugin requires no additional configuration as part of this specification. However this specification reserves all `PropertyQos` names with the prefix "***dds.sec.crypto.***" for use in future revisions of this specification.

9.5.2 DDS:Crypto:AES-GCM-GMAC Types

The `Cryptographic` plugin defines a set of generic data types to be used to initialize the plugin and to externalize the properties and material that must be shared with the applications that need to decode the cipher material, verify signatures, etc.

Each plugin implementation defines the contents of these types in a manner appropriate for the algorithms it uses. All "Handle" types are local opaque handles that are only understood by the local plugin objects that create or use them. The remaining types shall be fully specified so that independent implementations of DDS:Crypto:AES-GCM-GMAC can interoperate.

9.5.2.1 DDS:Crypto:AES-GCM-GMAC CryptoToken

The DDS:Crypto:AES-GCM-GMAC plugin shall set the attributes of the `CryptoToken` object as specified in the table below:

Table 66 – CryptoToken class for the builtin Cryptographic plugin

Attribute name	*Attribute value*	
class_id	"DDS:Crypto:AES_GCM_GMAC"	
binary_properties	*name*	*value*
	dds.cryp.keymat	The Big Endian CDR Serialization of the `KeyMaterial_AES_GCM_GMAC` structure defined below.

9.5.2.1.1 KeyMaterial_AES_GCM_GMAC structure

The contents and serialization of the `KeyMaterial_AES_GCM_GMAC` structure are described by the Extended IDL below.

Note: The types `CryptoTransformationKind` and `CryptoTransformKeyId` were defined in section 8.5.1.5.

```
/* Valid values for CryptoTransformKind  */

/* No encryption, no authentication tag */
#define CRYPTO_TRANSFORMATION_KIND_NONE              {0, 0, 0, 0}

/* No encryption.
   One AES128-GMAC authentication tag using the sender_key
   Zero or more AES128-GMAC auth. tags with receiver specfic keys */
#define CRYPTO_TRANSFORMATION_KIND_AES128_GMAC       {0, 0, 0, 1}

/* Authenticated Encryption using AES-128 in Galois Counter Mode
   (GCM) using the sender key.
   The authentication tag using the sender_key obtained from GCM
   Zero or more AES128-GMAC auth. tags with receiver specfic keys */
#define CRYPTO_TRANSFORMATION_KIND_AES128_GCM        {0, 0, 0, 2}

/* No encryption.
   One AES256-GMAC authentication tag using the sender_key
   Zero or more AES256-GMAC auth. tags with receiver specfic keys */
#define CRYPTO_TRANSFORMATION_KIND_AES256_GMAC       {0, 0, 0, 3}

/* Authenticated Encryption using AES-256 in Galois Counter Mode
   (GCM) using the sender key.
   The authentication tag using the sender_key obtained from GCM
   Zero or more AES256-GMAC auth. tags with receiver specfic keys */
#define CRYPTO_TRANSFORMATION_KIND_AES256_GCM        {0, 0, 0, 4}


@extensibility(FINAL)
struct KeyMaterial_AES_GCM_GMAC {
    CryptoTransformKind  transformation_kind;
    sequence<octet, 32>  master_salt;

    CryptoTransformKeyId sender_key_id;
    sequence<octet, 32>  master_sender_key;

    CryptoTransformKeyId receiver_specific_key_id;
    sequence<octet, 32>  master_receiver_specific_key;
};

typedef
sequence<KeyMaterial_AES_GCM_GMAC> KeyMaterial_AES_GCM_GMAC_Seq;
```

A zero value for *receiver_specific_key_id* indicates there is no receiver-specific authentication tags and shall occur if and only if the length of the *master_receiver_specific_key* is also zero.

9.5.2.1.2 Key material used by the BuiltinParticipantVolatileMessageSecureWriter and BuiltinParticipantVolatileMessageSecureReader

The Key Material used by the ***BuiltinParticipantVolatileMessageSecureWriter*** and ***BuiltinParticipantVolatileMessageSecureReader*** shall be derived from the `SharedSecret` obtained as part of the authentication process. The attributes of the `KeyMaterial_AES_GCM_GMAC` shall be set as described in Table 67 below. This uses HMAC-Based Key Derivation (HKDF) recommended in IETF RFC 5869 [50].

Table 67 – KeyMaterial_AES_GCM_GMAC for BuiltinParticipantVolatileMessageSecureWriter and BuiltinParticipantVolatileMessageSecureReader

Attribute name	Attribute value
transformation_kind	CRYPTO_TRANSFORMATION_KIND_AES256_GCM
master_salt	HMACsha256 (sha256(Challenge1 | KxSaltCookie | Challenge2) , SharedSecret) The parameters to the above functions are defined in Table 68. In the case where transformation_kind is CRYPTO_TRANSFORMATION_KIND_AES128_GCM this is truncated to the first 128 bits.
sender_key_id	0
master_sender_key	HMACsha256 (sha256(Challenge2 | KxKeyCookie | Challenge1) , SharedSecret) The parameters to the above functions are defined in Table 68. In the case where transformation_kind is CRYPTO_TRANSFORMATION_KIND_AES128_GCM this is truncated to the first 128 bits.
receiver_specific_key_id	0
master_receiver_specific_key	Zero-length sequence

Table 68 – Terms used in KxKey and KxMacKey derivation formula for the builtin Cryptographic plugin

Term	Meaning
Challenge1	The challenge that was sent in the ***challenge1*** attribute of the `HandshakeRequestMessageToken` as part of the Authentication protocol. This information shall be accessible from the `SharedSecretHandle`.

Challenge2	The challenge that was sent in the *challenge2* attribute of the `HandshakeReplyMessageToken` as part of the Authentication protocol.
	This information shall be accessible from the `SharedSecretHandle`.
SharedSecret	The shared secret established as part of the key agreement protocol.
	This information shall be accessible from the `SharedSecretHandle`.
KxKeyCookie	The 16 bytes in the string "key exchange key"
KxSaltCookie	The 16 bytes in the string "keyexchange salt"
data1 | data2 | data3	The symbol '|' is used to indicate byte string concatenation
HMACsha256(key, data)	Computes the hash-based message authentication code on 'data' using the key specified as first argument and a SHA256 hash as defined in [27].

9.5.2.2 DDS:Crypto:AES-GCM-GMAC CryptoTransformIdentifier

The DDS:Crypto:AES-GCM-GMAC shall set the `CryptoTransformIdentifier` attributes as specified in the table below:

Table 69 – CryptoTransformIdentifier class for the builtin Cryptographic plugin

Attribute	*Value*
`transformation_kind`	Set to one of the following values (see section 9.5.2.1.1):

	`CRYPTO_TRANSFORMATION_KIND_NONE`
	`CRYPTO_TRANSFORMATION_KIND_AES128_GMAC`
	`CRYPTO_TRANSFORMATION_KIND_AES128_GCM`
	`CRYPTO_TRANSFORMATION_KIND_AES256_GMAC`
	`CRYPTO_TRANSFORMATION_KIND_AES256_GCM`

	The variants containing AES128 in their name indicate that the encryption and/or authentication use AES with 128-bit key as the underlaying cryptographic engine. These variants shall have *master_sender_key* with 16 octets in length and *master_receiver_specific_key* with either zero or 16 octets in length.
	The variants containing AES256 in their name indicate that the encryption and/or authentication use AES with 256-bit key as the underlaying cryptographic engine. These variants shall have *master_sender_key* with 32 octets in length and *master_receiver_specific_key* with either zero or 32 octets in

	length.
	The variants with name ending with GCM indicate that the transformation is the standard authenticated encryption operation known as AES-GCM (AES using Galois Counter Mode) where the plaintext is encrypted and followed by an authentication tag computed using the same secret key. These variants may contain zero or more receiver-specific authentication tags. If *receiver_specific_key_id* is set to zero there shall be no receiver-specific tags otherwise there shall be one or more receiver-specific tags.
	The variants ending in GMAC indicate that there is no encryption (i.e., the *ciphertext* matches the input *plaintext*) and there is an authentication tag computed using the sender key that is shared with all the readers. These variants may contain zero or more receiver-specific authentication tags. If *receiver_specific_key_id* is set to zero there shall be no receiver-specific tags otherwise there shall be one or more receiver-specific tags.
`transformation_key_id`	This is set to a different value each time new Key Material is produced by a `DomainParticipant`. The algorithm used is implementation specific but it shall avoid repeating the values for the same `DomainParticipant`.

9.5.2.3 DDS:Crypto:AES-GCM-GMAC CryptoHeader

The DDS:Crypto:AES-GCM-GMAC `CryptoTransform` interface has several operations that transform `plain` text into `cipher` text. The cipher-text created by these "encode" operations contains a `CryptoHeader` that is interpreted by the corresponding "decode" operations on the receiving side.

The `CryptoHeader` structure is described by the Extended IDL below:

```
// Serialized as Big Endian
@extensibility(FINAL)
struct CryptoHeader {
    CryptoTransformIdentifier transform_identifier;
    octet                     session_id[4];
    octet                     initialization_vector_suffix[8];
};
```

As indicated by the IDL above, the *plugin_crypto_header_extra* attribute introduced in section 7.3.6.4.2 consists of the *session_id* and the *initialization_vector_suffix*.

The *transformation_indentifier* combined with the identity of the sending `DomainParticipant` uniquely identifies the `KeyMaterial` used to transform the plaintext into the `cipher` text.

The *session_id* combined with the `KeyMaterial` uniquely identifies the cryptographic keys used for the encryption and MAC operations.

The *initialization_vector_suffix* combined with the *session_id* uniquely identifies the
`Initialization Vector` used as part of the AES-GCM and AES-GMAC transformations.

The `CryptoHeader` structure shall be serialized using Big Endian serialization (a.k.a. network byte order).

9.5.2.4 DDS:Crypto:AES-GCM-GMAC CryptoContent

The DDS:Crypto:AES-GCM-GMAC `CryptoTransform` interface has operations that transform plaintext into `cipher` text. The cipher-text created by some of these "encode" operations contains a `CryptoContent` submessage element (see 7.3.6.2) that is interpreted by the corresponding "decode" operations on the receiving side.

The `CryptoContent` structure is described by the Extended IDL below:

```
// Serialized as Big Endian
@extensibility(FINAL)
struct CryptoContent  {
    sequence<octet>     crypto_content;
};
```

The `CryptoContent` structure shall be serialized using Big Endian serialization (a.k.a. network byte order).

9.5.2.5 DDS:Crypto:AES-GCM-GMAC CryptoFooter

The DDS:Crypto:AES-GCM-GMAC `CryptoTransform` interface has several operations that transform plaintext into `cipher` text. The cipher-text created by these "encode" operations contains a `CryptoFooter` that is interpreted by the corresponding "decode" operations on the receiving side.

The `CryptoFooter` structure is described by the Extended IDL below:

```
// Serialized as Big Endian
@extensibility(FINAL)
struct ReceiverSpecificMAC {
    CryptoTransformKeyId  receiver_mac_key_id;
    octet                 receiver_mac[16];
};

// Serialized as Big Endian
@extensibility(FINAL)
struct CryptoFooter {
    octet                             common_mac[16];
    sequence<ReceiverSpecificMAC>  receiver_specific_macs;
};
```

As indicated by the IDL above, the *crypto_footer* attribute introduced in section 7.3.6.5 consists of the *common_mac* and the *receiver_specific_macs*.

The receiver-specific Message Authentication Codes (MACs) are computed with a secret key that the sender shares only with one receiver. The receiver-specific MACs provide message-origin

authentication to the receiver even when the sender is communicating with multiple receivers via multicast and shares the same encryption key will all of them.

The `ReceiverSpecificMAC` and `CryptoFooter` structures shall be serialized using Big Endian serialization (a.k.a. network byte order).

9.5.3 DDS:Crypto:AES-GCM-GMAC plugin behavior

This plugin implements three interfaces: `CryptoKeyFactory`, `CryptoKeyExchange`, and `CryptoTransform`. Each is described separately.

9.5.3.1 CryptoKeyFactory for DDS:Crypto:AES-GCM-GMAC

The table below describes the actions that the DDS:Crypto:AES-GCM-GMAC when each of the `CryptoKeyFactory` plugin operations is invoked.

Table 70 – Actions undertaken by the operations of the builtin Cryptographic CryptoKeyFactory plugin

`register_local_pa rticipant`	This operation shall create a new `KeyMaterial_AES_GCM_GMAC` object and return a handle that the plugin can use to access the created object. We will refer to this object by the name: `ParticipantKeyMaterial`.
	The ***transformation_kind*** for the `ParticipantKeyMaterial` object determines whether the transformation performs authentication only (GMAC) or authenticated encryption (GCM). The selection between these two options shall be done according to the setting of the RTPS Protection Kind (see 9.4.1.2.5.6).
	The ***transformation_kind*** also determines whether the encryption and/or authentication uses 128-bit or 256-bit keys. This aspect shall be configurable but the configuration mechanism is not specified.
`register_matched_ remote_participan t`	This operation shall associate the `SharedSecret` received as an argument with the local and remote `ParticipantCryptoHandle`.
	This operation shall create a new `KeyMaterial_AES_GCM_GMAC` object and associate it with the local and remote `ParticipantCryptoHandle` pair. We will refer to this object by the name: `Participant2ParticipantKeyMaterial`.
	The `Participant2ParticipantKeyMaterial` ***transformation_kind***, ***master_salt***, and ***master_sender_key, and sender_key_id*** shall match those of the `ParticipantKeyMaterial`.
	If the RTPS Protection Kind (see 9.4.1.2.5.6) does not specify the use of origin authentication, then the ***receiver_specific_key_id*** shall be set to zero and the ***master_receiver_specific key*** shall be set to the empty sequence.

	If the RTPS Protection Kind (see 9.4.1.2.5.6) specifies the use of origin authentication, then a new secret key (`MasterReceiverParticipantSpecificKey`) shall be created, the *receiver_specific_key_id* shall be set to identify this new key, and the *master_receiver_specific key* field shall contain `MasterReceiverParticipantSpecificKey.`
	The `Participant2ParticipantKeyMaterial` shall be used to transform and authenticate the RTPS messages.
	The `Participant2ParticipantKeyMaterial` shall be sent to the remote DomainParticipant using the operations of the `CryptoKeyExchange`.
	This operation also creates a `KeyMaterial_AES_GCM_GMAC` object derived from the `SharedSecret` passed as a parameter. This key material shall be associated with the local and remote `ParticipantCryptoHandle` pair. We will refer to this key material as the `Participant2ParticipantKxKeyMaterial`. It is used to exchange key material between `DomainParticipant` entities.
`register_local_da tawriter`	This operation shall create a new `KeyMaterial_AES_GCM_GMAC_Seq` object and returns a handle that the plugin can use to access the created object. We will refer to this object by the name: `WriterKeyMaterialSeq`. The sequence may contain one or two elements depending on the settings of the Metadata Protection Kind (see 9.4.1.2.6.6) and Data Protection Kind (see 9.4.1.2.6.7).
	If the Metadata Protection Kind is different from NONE, then the operation shall create a `KeyMaterial_AES_GCM_GMAC` to use for the `encode_datawriter_submessage` operation. In addition, this key material shall appear as the first element in the `KeyMaterial_AES_GCM_GMAC_Seq`.
	If the Data Protection Kind is different from NONE, then the operation shall create a `KeyMaterial_AES_GCM_GMAC` to use for the `encode_serialized_payload` operation.
	In the case where both meta-data protection and data protection are the same, it is allowed for an implementation to reuse the same key material for both. In this case the `KeyMaterial_AES_GCM_GMAC_Seq` would contain only one element. This "key reuse" aspect shall be configurable but the configuration mechanism is not specified.
	The *transformation_kind* for the `KeyMaterial_AES_GCM_GMAC` objects determines whether the transformation performs authentication only (GMAC) or encryption followed by authentication (GCM). The

	selection between these two options for each created `KeyMaterial_AES_GCM_GMAC` object shall be done according to the setting of corresponding Protection Kind.

The ***transformation_kind*** for the `KeyMaterial_AES_GCM_GMAC` objects also determines whether the encryption and/or authentication uses 128-bit or 256-bit keys. This aspect shall be configurable but the configuration mechanism is not specified. |
| `register_matched_ remote_datareader` | This operation shall create a new `KeyMaterial_AES_GCM_GMAC_Seq` object and associate it with the local `DatawriterCryptoHandle` and remote `DatareaderCryptoHandle` pair. We will refer to this object by the name: `Writer2ReaderKeyMaterialSeq`.

The first elements of the `Writer2ReaderKeyMaterialSeq` shall contain the elements of the `WriterKeyMaterialSeq`.

Additional elements depend on whether the Metadata Protection Kind (see 9.4.1.2.6.6) specified the use of origin authentication.

If the Metadata Protection Kind (see 9.4.1.2.6.6) specified the use of origin authentication, the first element of the `Writer2ReaderKeyMaterialSeq` shall contain a non-zero ***receiver_specific_key_id*** that identifies a new key created by this operation. The new key (`MasterReceiverREndpointSpecificKey`) shall be stored in the ***master_receiver_specific_key***. This ***master_receiver_specific_key*** shall be shared only with that one specific remote `DataReader` so that it can be used to authenticate the `DataWriter` that originated the message.

The `Writer2ReaderKeyMaterialSeq` shall be sent to the remote DataReader such that it can process the `CryptoTransform` encoded from the `DataWriter`. |
| `register_local_da tareader` | This operation shall create a new `KeyMaterial_AES_GCM_GMAC` object and return a handle that the plugin can use to access the created object. We will refer to this object by the name: `ReaderKeyMaterial`.

The ***transformation_kind*** for the `ReaderKeyMaterial` object determines whether the transformation performs authentication only (GMAC) or encryption followed by authentication (GCM). The selection between these two options shall be done according to the setting of the Data Protection Kind (see 9.4.1.2.6.7).

The ***transformation_kind*** also determines whether the encryption and/or authentication uses 128-bir or 256-bit keys. This aspect shall be configurable but the configuration mechanism is not specified. |

`register_matched_ remote_datawriter`	This operation shall create a new `KeyMaterial_AES_GCM_GMAC` object and associate it with the local `DatareaderCryptoHandle` and remote `DatawriterCryptoHandle` pair. We will refer to this object by the name: `Reader2WriterKeyMaterial`. The ***transformation_kind***, ***master_salt***, and ***master_sender_key, and sender_key_id*** for the `Reader2WriterKeyMaterial` object shall match those in the `DataReader ReaderKeyMaterial`. If the Metadata Protection Kind (see 9.4.1.2.6.6) does not specify the use of origin authentication, then the ***receiver_specific_key_id*** shall be set to zero and the ***master_receiver_specific key*** shall be set to the empty sequence. If the Metadata Protection Kind (see 9.4.1.2.6.6) specifies the use of origin authentication, then a new secret key (`MasterReceiverWEndpointSpecificKey`) shall be created, the ***receiver_specific_key_id*** shall be set to identify this new key, and the ***master_receiver_specific key*** field shall contain `MasterReceiverWEndpointSpecificKey`. The `Reader2WriterKeyMaterial` shall be sent to the remote `DataWriter` such that it can process the `ciphetext` from the `DataReader`.
`unregister_partic ipant`	Releases any resources allocated on the corresponding call to `register_local_participant`, or `register_matched_remote_participant`.
`unregister_datawr iter`	Releases any resources allocated on the corresponding call to `register_local_datawriter`, or `register_matched_remote_datawriter`.
`unregister_datare ader`	Releases any resources allocated on the corresponding call to `register_local_datareader`, or `register_matched_remote_datareader`.

9.5.3.2   CryptoKeyExchange for DDS:Crypto:AES-GCM-GMAC

The table below describes the actions that the DDS:Crypto:AES-GCM-GMAC when each of the `CryptoKeyExchange` plugin operations is invoked.

Table 71 – Actions undertaken by the operations of the builtin Cryptographic CryptoKeyExchange plugin

`create_local_part icipant_crypto_to kens`	Creates a DDS:Crypto:AES-GCM-GMAC `CryptoToken` object and returns it in the output sequence. The `CryptoToken` contains the `Participant2ParticipantKeyMaterial` created on the call to `register_matched_remote_participant` for the `remote_participant_crypto`.

`set_remote_partic ipant_crypto_toke ns`	Shall receive the sequence containing one `CryptoToken` object that was created by the corresponding call to `create_local_participant_crypto_tokens` on the remote side.
`create_local_data writer_crypto_tok ens`	Creates a DDS:Crypto:AES-GCM-GMAC `CryptoToken` object and returns it in the output sequence. The `CryptoToken` contains the `Writer2ReaderKeyMaterial` created on the call to `register_matched_remote_datareader` for the `remote_datareader_crypto`.
`set_remote_datawr iter_crypto_token s`	Shall receive the sequence containing one `CryptoToken` object that was created by the corresponding call to `create_local_datawriter_crypto_tokens` on the remote side.
`create_local_data reader_crypto_tok ens`	Creates a DDS:Crypto:AES-GCM-GMAC `CryptoToken` object and returns it in the output sequence. The `CryptoToken` contains the `Reader2WriterKeyMaterial` created on the call to `register_matched_remote_datawriter` for the `remote_datawriter_crypto`.
`set_remote_datare ader_crypto_token s`	Shall receive the sequence containing one `CryptoToken` object that was created by the corresponding call to `create_local_datareader_crypto_tokens` on the remote side.
`return_crypto_tok ens`	Releases the resources associated with the `CryptoToken` objects in the sequence.

9.5.3.3   CryptoKeyTransform for DDS:Crypto:AES-GCM-GMAC

9.5.3.3.1   Overview

The table below describes the actions that the DDS:Crypto:AES-GCM-GMAC when each of the `CryptoKeyTransform` plugin operations is invoked.

Table 72 – Actions undertaken by the operations of the builtin Cryptographic CryptoKeyTransform plugin

`encode_serializ ed_payload`	Uses the `WriterKeyMaterial` associated with the `sending_datawriter_crypto` to encrypt and/or sign the input `SerializedPayload` RTPS SubmessageElement (see 7.3.1). If the *transformation_kind* indicates that encryption is performed, then the output shall be the three RTPS Submessage elements:

	CryptoHeader, CryptoContent, and CryptoFooter (see 9.4.2.5 and 9.5.3.3.4.4).

If the *transformation_kind* indicates that only authentication is performed, then the output shall be the three RTPS Submessage elements: CryptoHeader, SerializedPayload, and CryptoFooter. Where SerializedPayload is the serialized payload passed as an input to the operation.

This operation shall always set the *receiver_specific_macs* attribute in the `CryptoFooter` to the empty sequence. |
| `encode_datawrit er_submessage` | Uses the `WriterKeyMaterial` associated with the `sending_datawriter_crypto` and the `Writer2ReaderKeyMaterial` associated with the `sending_datawriter_crypto` and each of the `receiving_datareader_crypto` handles to encrypt and/or sign the input RTPS Submessage.

If the *transformation_kind* indicates that encryption is performed, then the output shall be the three RTPS Submessages: `SecurePrefixSubMsg`, `SecureBodySubMsg`, and `SecurePostfixSubMsg`. See 7.3.7.6, 7.3.7.5, and 7.3.7.7.

If the *transformation_kind* indicates that only authentication is performed, then the output shall be the three RTPS Submessages: `SecurePrefixSubMsg`, `InputSubmessage`, and `SecurePostfixSubMsg`. Where `InputSubmessage` indicates the submessage that was passed as input to the operation.

The transformations shall be computed using the `WriterKeyMaterial` associated with the `sending_datawriter_crypto`.

Depending on the configuration the operation may compute and set the *common_mac* and the *receiver_specific_macs* attributes within the `SecurePostfixSubMsg`.

The *common_mac* shall be computed using the `WriterKeyMaterial` associated with the `sending_datawriter_crypto`.

If computed, the *receiver_specific_macs* shall be computed using the `Writer2ReaderKeyMaterial` associated with the pair composed of the `sending_datawriter_crypto` and each of the corresponding `receiving_datareader_crypto`.

In the case of ***BuiltinParticipantVolatileMessageSecureWriter***, the ***receiving_datareader_crypto_list*** has ONE element containing `KxKey` material derived from the `SharedSecret` as described in 9.5.2.1.2. |

`encode_datareader_submessage`	Uses the `ReaderKeyMaterial` associated with the `sending_datareader_crypto` and the `Reader2WriterKeyMaterial` associated with the `sending_datareader_crypto` and each of the `receiving_datareader_crypto` handles to encrypt and/or sign the input RTPS Submessage.

If the ***transformation_kind*** indicates that encryption is performed, then the output shall be the three RTPS Submessages: `SecurePrefixSubMsg`, `SecureBodySubMsg`, and `SecurePostfixSubMsg`. See 7.3.7.6, 7.3.7.5, and 7.3.7.7.

If the ***transformation_kind*** indicates that only authentication is performed, then the output shall be the three RTPS Submessages: `SecurePrefixSubMsg`, `InputSubmessage`, and `SecurePostfixSubMsg`. Where `InputSubmessage` indicates the submessage that was passed as input to the operation.

The transformations shall be computed using the `ReaderKeyMaterial` associated with the `sending_datareader_crypto`.

Depending on the configuration the operation may compute and set the common_digest or the additional_digests.

The ***common_mac*** shall be computed using the `ReaderKeyMaterial` associated with the `sending_datareader_crypto`.

If computed, the ***receiver_specific_macs*** shall be computed using the `Reader2WriterKeyMaterial` associated with the pair composed of the `sending_datareader_crypto` and each of the corresponding `receiving_datawriter_crypto`.

In the case of ***BuiltinParticipantVolatileMessageSecureReader***, the ***receiving_datawriter_crypto_list*** has ONE element containing `KxKey` material derived from the `SharedSecret` as described in 9.5.2.1.2. |

`encode_rtps_message`	Transforms the input RTPS Message into an output RTPS Message that contains the original RTPS Header followed by the `SecureRTPSPrefixSubMsg`, one or more RTPS SubMessages, and the `SecureRTPSPostfixSubMsg`.
	The transformation uses the `ParticipantKeyMaterial` associated with the `sending_participant_crypto` and `Participant2ParticipantKeyMaterial` and each of the `receiving_participant_crypto` handles.
	Let `RTPSMessage{RTPSHdr-> InfoSourceSubMsg}` represent the input RTPS Message transformed so that the RTPS Header is replaced with an RTPS InfoSourceSubMsg containing the same information as the RTPS Header and the remaining submessages remain the same.
	If the *transformation_kind* indicates that encryption is performed, then the output shall be the three RTPS Submessages: `SecureRTPSPrefixSubMsg`, `SecureBodySubMsg`, and `SecureRTPSPostfixSubMsg`.
	The `SecureBodySubMsg` shall contain the result of encrypting the `RTPSMessage{RTPSHdr-> InfoSourceSubMsg}`.
	The `SecureRTPSPostfixSubMsg` shall contain the authentication tags computed on the `SecureBodySubMsg`.
	If the *transformation_kind* indicates that only authentication is performed then the output shall be the RTPS Submessages: `SecureRTPSPostfixSubMsg`, `RTPSMessage{RTPSHdr-> InfoSourceSubMsg}`, and `SecureRTPSPostfixSubMsg`.
	The `SecureRTPSPostfixSubMsg` shall contain the authentication tags computed on the `SecurePrefixSubMsg`, `RTPSMessage{RTPSHdr-> InfoSourceSubMsg}`.
	Depending on the configuration the operation may contain only the *common_mac* and a non-zero length *receiver_specific_macs.*
	The *common_mac* shall be computed using the `ParticipantKeyMaterial` associated with the `sending_participant_crypto`.
	If present, the *receiver_specific_macs* shall be computed using the `Participant2ParticipantKeyMaterial` associated with the pair composed of the `sending_participant_crypto` and each of the corresponding `receiving_participant_crypto`.

`decode_rtps_message`	Examines the `SecureRTPSPrefixSubMsg` to determine the *transformation_kind* is one of the recognized kinds. If the kind is not recognized, the operation shall fail with an exception.

Uses source and destination `DomainParticipant` GUIDs in the RTPS Header to locate the `sending_participant_crypto` and `receiving_participant_crypto`. Then looks whether the *transformation_key_id* attribute in the `CryptoTransformIdentifier` is associated with those `ParticipantCryptoHandles`. If the association is not found the operation shall fail with an exception.

Uses the `RemoteParticipantKeyMaterial` and the `RemoteParticipant2ParticipantKeyMaterial` associated with the retrieved `ParticipantCryptoHandles` to validate the authentication tags contained in the `SecureRTPSPostfixSubMsg`. If the `RemoteParticipant2ParticipantKeyMaterial` specified a *receiver_specific_key_id* different from zero, the operation shall check that the received `SecureRTPSPostfixSubMsg` contains a *receiver_specific_macs* element containing the *receiver_specific_key_id* associated with local and remote `CryptoHandles` and use it to verify the submessage. If the *receiver_specific_key_id* is missing or the verification fails the operation shall fail with an exception.

Upon success the returned RTPS Message shall match the input to the `encode_rtps_message` operation on the DomainParticipant that sent the message. |
| `preprocess_secure_submsg` | Examines the RTPS SecureSubmessage to:

1. Determine whether the `CryptoTransformIdentifier` the *transformation_kind* matches one of the recognized kinds.
2. Classify the RTPS Submessage as a Writer or Reader Submessage.
3. Retrieve the DatawriterCryptoHandle and DataReaderCryptoHandle handles associated with the `CryptoTransformIdentifier` *transformation_key_id*. |

`decode_datawrit er_submessage`	Uses the `RemoteDatawriterKeyMaterial` and the `RemoteDatawriter2DatareaderKeyMaterial` associated with the CryptoHandles returned by the `preprocess_secure_submessage` to verify and decrypt the RTPS SubMessage that follows the `SecurePrefixSubMsg`, using the authentication tags in the `SecurePostfixSubMsg`. If the verification or decryption fails, the operation shall fail with an exception.
	If the `RemoteDatawriterKeyMaterial` specified a *transformation_kind* different from CRYPTO_TRANSFORMATION_KIND_NONE, then the operation shall check that the received `SecurePostfixSubMsg` contains a *common_mac* and use it to verify the RTPS SubMessage that follows the `SecurePrefixSubMsg`. If the *common_mac* is missing or the verification fails the operation shall fail with an exception.
	If the `RemoteDatawriter2DatareaderKeyMaterial` specified a *receiver_specific_key_id* different from zero, then the operation shall check that the received `SecurePostfixSubMsg` contains a non-zero length *receiver_specific_macs* element containing the *receiver_specific_key_id* that is associated with local and remote CryptoHandles and use it to verify the submessage. If the *receiver_mac_key_id* is missing or the verification fails, the operation shall fail with an exception.
	If the `RemoteDatawriterKeyMaterial` specified a *transformation_kind* that performs encryption the operation shall use the `RemoteDatawriterKeyMaterial` to decode the data in the `SecureBodySubMsg`, obtain an RTPS SubMessage and return it. Otherwise the RTPS Submessage that follows the `SecurePrefixSubMsg` is returned.
	Upon success the returned RTPS SubMessage shall match the input to the `encode_datawriter_message` operation on the DomainParticipant that sent the message.
	In the case of ***BuiltinParticipantVolatileMessageSecureReader***, the ***sending_datawriter_crypto*** contains the `KxKey` material derived from the `SharedSecret` as described in 9.5.2.1.2
`decode_dataread er_submessage`	Uses the `RemoteDatareaderKeyMaterial` and the `RemoteDatareader2DatawriterKeyMaterial` associated with the CryptoHandles returned by the `preprocess_secure_submessage` to verify and decrypt the RTPS SubMessage that follows the `SecurePrefixSubMsg`, using the authentication tags in the `SecurePostfixSubMsg`. If the verification or decryption fails, the operation shall fail with an exception.

	If the `RemoteDatareaderKeyMaterial` specified a *transformation_kind* different from CRYPTO_TRANSFORMATION_KIND_NONE, then the operation shall check that the received `SecurePostfixSubMsg` contains a *common_mac* and use it to verify the RTPS SubMessage that follows the `SecurePrefixSubMsg`. If the *common_mac* is missing or the verification fails, the operation shall fail with an exception.
	If the `RemoteDatareader2DatawriterKeyMaterial` specified a *receiver_specific_key_id* different from zero, then the operation shall check that the received `SecurePostfixSubMsg` contains a non-zero length *receiver_specific_macs* element containing the *receiver_specific_key_id* that is associated with local and remote CryptoHandles and use it to verify the submesage. If the *receiver_specific_key_id* is missing or the verification fails, the operation shall fail with an exception.
	If the `RemoteDatareaderKeyMaterial` specified a *transformation_kind* that performs encryption the operation shall use the `RemoteDatareaderKeyMaterial` to decode the data in the `SecureBodySubMs`, obtain an RTPS SubMessage and return it. Otherwise the RTPS Submessage that follows the `SecurePrefixSubMsg` is returned.
	Upon success the returned RTPS SubMessage shall match the input to the `encode_datareader_message` operation on the DomainParticipant that sent the message.
	In the case of ***BuiltinParticipantVolatileMessageSecureWriter***, the *sending_datareader_crypto* contains the `KxKey` material derived from the `SharedSecret` as described in 9.5.2.1.2
`decode_serializ ed_payload`	Uses writerGUID and the readerGUID in the RTPS SubMessage to locate the `sending_datawriter_crypto` and `receiving_datareader_crypto`. Then looks whether the *transformation_key_id* attribute in the `CryptoTransformIdentifier` in the `CryptoHeader` SubmessageElement is associated with those CryptoHandles. If the association is not found, the operation shall fail with an exception.
	Uses the `RemoteDatawriterKeyMaterial` associated with the retrieved CryptoHandles to verify the *common_mac* and decrypt the RTPS SecureData SubmessageElement. If the verification or decryption fails, the operation shall fail with an exception.
	If the `RemoteDatawriterKeyMaterial` specified a *receiver_specific_key_id* different from zero, then the operation shall check that the received SecureData SubmessageElement contains a non-zero length *receiver_specific_macs* element containing the

	receiver_specific_key_id that is associated with the local and remote CryptoHandles. If the *receiver_specific_key_id* is missing or the verification fails, the operation shall fail with an exception.
	If the `RemoteDatawriterKeyMaterial` specified a *transformation_kind* that performs encryption, the operation shall use the `RemoteDatawriterKeyMaterial` to decode the data in the `CryptoContent`, obtain a `SerializedPayload` and return it. Otherwise the RTPS Submessage Element that follows the `CryptoHeader` is returned as `SerializedPayload`.
	Upon success the returned RTPS `SerializedPayload` shall match the input to the `encode_serialized_payload` operation on the DomainParticipant that sent the message.

9.5.3.3.2 Encode/decode operation virtual machine

The logical operation of the DDS:Crypto:AES-GCM-GMAC is described in terms of a virtual machine as it performs the encrypt message digest operations. This is not intended to mandate implementations should follow this approach literally, simply that the observable results for any plaintext are the same as the virtual machine described here.

For any given cryptographic session the operation of the DDS:Crypto:AES-GCM-GMAC transforms plaintext into ciphertext can be described in terms of a virtual machine that maintains the following state:

Table 73 – Terms used in Key Computation and cryptographic transformations formulas for the builtin cryptographic plugin

State variable	Type	Meaning
MasterKey	128 bit array for AES128 256 bit array for AES256	The master key from which session salts, session keys and session hash keys are derived.
MasterSalt	128 bit array for AES128 256 bit array for AES256	A random vector used in connection with the MasterKey to create the SessionKey.
MasterKeyId	octet[4]	A NONCE value associated with the master key when it is first created used to tag the ciphertext to ensure the correct key is being used during decryption. It may be used also for the purposes of re-keying.
MasterReceiverSpecificKey	128 bit array for AES128 256 bit array for AES256	The master key from which SessionReceiverSpecificKey keys are derived.
InitializationVectorSuffix	octet[8]	An initially random NONCE used to

		create the Initialization Vector needed by the cryptographic operations. This value shall be changed each time an encryption or MAC operation is performed using the same key.
SessionId	octet[4]	An initially random value used to create the current SessionKey, and SessionReceiverSpecificKey from the MasterKey, MasterReceiverSpecificKey, and Master salts. The SessionId is incremented each time a new SessionKey is needed and then used to derive the new SessionKey and SessionReceiverSpecificKey from the MasterKey and MasterReceiverSpecificKey. Knowledge of the MasterKey, MasterSalt, and the SessionId is sufficient to create the SessionKey. Knowledge of the MasterReceiverSpecificKey, MasterSalt, and the SessionId is sufficient to create the SessionReceiverSpecificKey.
SessionKey	128 bit array for AES128 256 bit array for AES256	The current key used for creating the ciphertext and/or the common_mac. It is constructed from the MasterKey, MasterSalt, and SessionId.
SessionReceiverSpecificKey	128 bit array for AES128 256 bit array for AES256	The current key used for creating the receiver_specific_mac.
session_block_counter	64 bit integer	A counter that counts the number of blocks that have been ciphered with the current SessionKey.
max_blocks_per_session	64 bit integer	A configurable property that limits the number of blocks that can be ciphered with the same SessionKey. If the *session_block_counter* exceeds this value a new SessionKey and SessionReceiverSpecificKey are computed and the *session_block_counter* is reset to zero.

All the key material with a name that starts with "Master" corresponds to the `KeyMaterial_AES_GCM_GMAC` objects that were created by the `CryptoKeyFactory` operations. This key material is not used directly to encrypt or compute MAC of the plaintext. Rather it is used to create "Session" Key material by means of the algorithms described below. This has the benefit that the 'session' keys used to secure the data stream data can be modified as needed to maintain the security of the stream without having to perform explicit rekey and key-exchange operations.

9.5.3.3.3   Computation of SessionKey and SessionReceiverSpecificKey

The `SessionKey` and `SessionReceiverSpecificKey` are computed from the `MasterKey`, `MasterSalt` and the `SessionId`:

```
SessionKey := HMAC256(MasterKey,"SessionKey" | MasterSalt | SessionId)

SessionReceiverSpecificKey
    := HMAC256(MasterReceiverSpecificKey,
               "SessionReceiverKey" | MasterSalt | SessionId)
```

HMAC256 is a HMAC-SHA256. In case a 128 key is desired the 256 bit HMAC is truncated to the first 128 bits.

In the above expressions the symbol '|' indicates concatenation.

9.5.3.3.4   Computation of ciphertext from plaintext

The ciphertext is computed from the plain text using AES in Galois Counter Mode (AES-GCM).

The encryption transforms the plaintext input into ciphertext by performing an encryption operation using the AES-GCM algorithm in counter mode using the SessionKeys associated with the specified KeyHandle. The encryption transformation is described in detail in the sections that follow.

The encryption operation uses a 96-bit initialization vector constructed as:

```
    InitializationVector =  SessionId | InitializationVectorSuffix
```

In the above expression '|' indicates the concatenation of bit strings.

The same *InitializationVector* is associated with all the session keys (*SessionKey* and all *SessionReceiverSpecificKeys*) associated with a specific Sender. It shall be incremented each time any of those keys are used to encrypt and/or create a MAC.

The *session_block_counter* is an internal counter that keeps track of the number of blocks encrypted with the same session key. The purpose is to ensure that a single session key is not used to encrypt more than the configured *max_blocks_per_session*. The *session_block_counter* and the size of the plain text shall be used by implementations of the `Crypto` encode operations to ensure that *max_blocks_per_session* will not be exceeded during the encode operation. If the operation detects that the counter would exceed the maximum then it should modify the *SessionId* and derive new session keys prior to transforming any of the input plain text. The change in the *SessionId* creates new session keys and thus resets the *session_block_counter*. This approach ensures that all ciphertext returned by the operation is encrypted with the same session keys.

The resulting ciphertext will be preceded by a `CryptoHeader` that indicates the *SessionId* and *InitializationVectorSuffix*.

The resulting block of bytes from the "encode" operations (`encode_serialized_payload`, `encode_datawriter_submessage`, `encode_datareader_submessage`, and `encode_rtps_message`) is illustrated in the sections that follow:

9.5.3.3.4.1   Format of the CryptoHeader Submessage Element

The `CryptoHeader` submessage element generated by the DDS:Crypto:AES-GCM-GMAC shall take the form:

```
0...2..........8................16.............24...............32
+--------------+--------------+--------------+--------------+
+ CryptoHeader:                                             +
+    CryptoTransformIdentifier  transformation_id           +
|      octet[4]   transformation_id.transformation_kind     |
|      octet[4]   transformation_id.transformation_key_id   |
+    - - - - - - - - - - - - - - - - - - - - - - - - - -    +
+    plugin_sec_prefix:                                     +
|      octet[4]   plugin_sec_prefix.session_id              |
~      octet[8]   plugin_sec_prefix.init_vector_suffix      ~
+--------------+--------------+--------------+--------------+
```

Note that as specified in subclause 0 the CryptoHeader shall be serialized using Big Endian representation.

9.5.3.3.4.2   Format of the CryptoContent Submessage Element

The `CryptoContent` submessage element generated by the DDS:Crypto:AES-GCM-GMAC shall take the form:

```
0...2..........8................16.............24...............32
+--------------+--------------+--------------+--------------+
+ CryptoContent:                                            +
|     long crypto_content.length = N                        |
+    - - - - - - - - - - - - - - - - - - - - - - - - - -    +
|crypto_ct[0]   |crypto_ct[1]   |crypto_ct[2]   |crypto_ct[3]   |
~               . . .                                       ~
|crypto_ct[N-4] |crypto_ct[N-3] |crypto_ct[N-2] |crypto_ct[N-1] |
+--------------+--------------+--------------+--------------+
```

Note that the cipher operations have 16-byte block-size and add padding when needed. Therefore the *secure data.length* ("N") will always be a multiple of 16.

Note that as specified in subclause 9.5.2.4 the *secure data.length* shall be serialized using Big Endian representation.

9.5.3.3.4.3   Format of the CryptoFooter Submessage Element

The `CryptoFooter` submessage element generated by the DDS:Crypto:AES-GCM-GMAC shall take the form:

```
0...2...........8................16...............24................32
+--------------+--------------+--------------+--------------+
+ CryptoFooter ( = plugin_sec_tag):                        +
~    octet[16] plugin_sec_tag.common_mac                    ~
+    - - - - - - - - - - - - - - - - - - - - - - - - -      +
+    plugin_sec_tag.receiver_specific_macs:                 +
|      long    plugin_sec_tag.receiver_specific_macs.length = N |
|      - - - - - - - - - - - - - - - - - - - - - - - - - -   |
|      octet[4]  receiver_specific_macs[0].receiver_mac_key_id  |
|      octet[16] receiver_specific_macs[0].receiver_mac     ~
+      - - - - - - - - - - - - - - - - - - - - - - - -      +
+      . . .                                                +
+      - - - - - - - - - - - - - - - - - - - - - - - -      +
|      octet[4]  receiver_specific_macs[N-1].receiver_mac_key_id|
~      octet[16] receiver_specific_macs[N-1].receiver_mac   ~
+--------------+--------------+--------------+--------------+
```

Note that as specified in subclause 9.5.2.5 the CryptoHeader shall be serialized using Big Endian representation.

9.5.3.3.4.4   Result from encode_serialized_payload

The input to this operation is a SerializedPayload submessage element:
```
0...2...........8................16.............24................32
+--------------+--------------+--------------+--------------+
~          SerializedPayload                                ~
+--------------+--------------+--------------+--------------+
```

The output in case the transformation performs authentication only shall be:
```
0...2...........8................16.............24................32
+--------------+--------------+--------------+--------------+
~          CryptoHeader                                     ~
+--------------+--------------+--------------+--------------+
~          SerializedPayload    (unchanged from input)      ~
+--------------+--------------+--------------+--------------+
~          CryptoFooter                                     ~
+--------------+--------------+--------------+--------------+
```

The *common_mac* in the `CryptoFooter` is the authentication tag generated by the AES-GMAC transformation using the *SessionKey* and the *InitializationVector* operating on the `SerializedPayload`.

The *receiver_specific_macs* in the `CryptoFooter` are the AES-GMAC tags computed on the *common_mac* using each of the *SessionReceiverSpecificKey* and the same *InitializationVector*.

The output in case the transformation performs encryption and authentication shall be:

```
0...2...........8................16..............24...............32
+--------------+--------------+--------------+--------------+
~        CryptoHeader                                      ~
+--------------+--------------+--------------+--------------+
~        CryptoContent                                     ~
|        crypto_content =  Encrypt(SerializedPayload)      |
+--------------+--------------+--------------+--------------+
~        CryptoFooter                                      ~
+--------------+--------------+--------------+--------------+
```
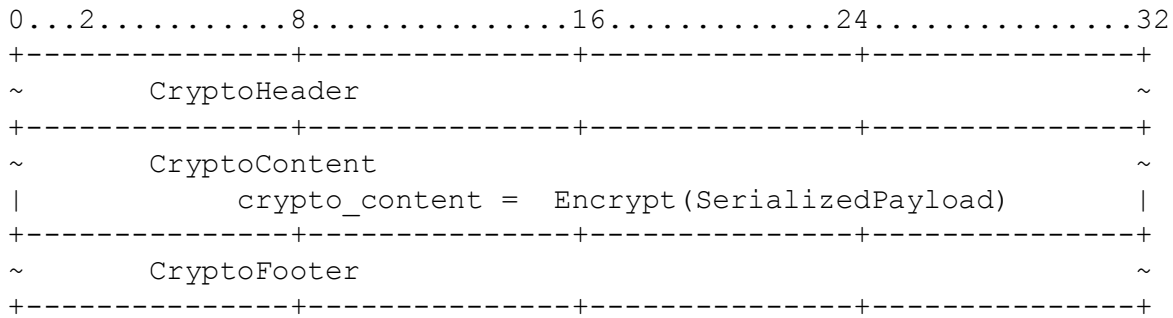
In the above Encrypt indicates the cryptographic transformation performed with AES-GCM using the *SessionKey* and the *InitializationVector* operating on the `SerializedPayload`.

The *common_mac* in the `CryptoFooter` is the authentication tag generated by the same AES-GCM where the Additional Authenticated Data is empty.

The *receiver_specific_macs* in the `CryptoFooter` are the AES-GMAC tags computed on the *common_mac* using each of the *SessionReceiverSpecificKey* and the same *InitializationVector*.

9.5.3.3.4.5   Result from encode_datawriter_submessage and encode_datareader_submessage

The input to this operation is an RTPS submessage:
```
0...2...........8................16..............24...............32
+--------------+--------------+--------------+--------------+
|                                                          |
~        RTPS SubMessage                                   ~
|                                                          |
+--------------+--------------+--------------+--------------+
```

The output in case the transformation performs authentication only shall be:
```
0...2...........8................16..............24...............32
+--------------+--------------+--------------+--------------+
| SEC_PREFIX   | (flags)     E|    short octetsToNextSubMsg |
+--------------+--------------+--------------+--------------+
~    CryptoHeader                                          ~
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
|                                                          |
~    RTPS SubMessage         (unchanged from input)        ~
|                                                          |
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SEC_POSTFIX  | (flags)     E|    short octetsToNextSubMsg |
+--------------+--------------+--------------+--------------+
~        CryptoFooter                                      ~
+--------------+--------------+--------------+--------------+
```

The *common_mac* in the `CryptoFooter` is the authentication tag generated by the AES-GMAC transformation using the *SessionKey* and the *InitializationVector* operating on the RTPS `Submessage`.

The *receiver_specific_macs* in the `CryptoFooter` are the AES-GMAC tags computed on the *common_mac* using each of the **SessionReceiverSpecificKey** and the same **InitializationVector**.

The output in case the transformation performs encryption and authentication shall be:

```
0...2..........8................16..............24...............32
+--------------+--------------+--------------+--------------+
| SEC_PREFIX   | (flags)     E|    short octetsToNextSubMsg  |
+--------------+--------------+--------------+--------------+
~      CryptoHeader                                         ~
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SEC_BODY     | (flags)     E|    short octetsToNextSubMsg  |
+--------------+--------------+--------------+--------------+
~      CryptoContent                                        ~
|        crypto_content = Encrypt( RTPS SubMsg )            |
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SEC_POSTFIX  | (flags)     E|    short octetsToNextSubMsg  |
+--------------+--------------+--------------+--------------+
~      CryptoFooter                                         ~
+--------------+--------------+--------------+--------------+
```
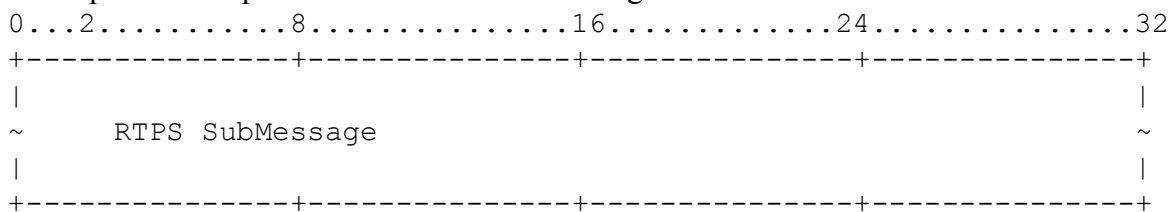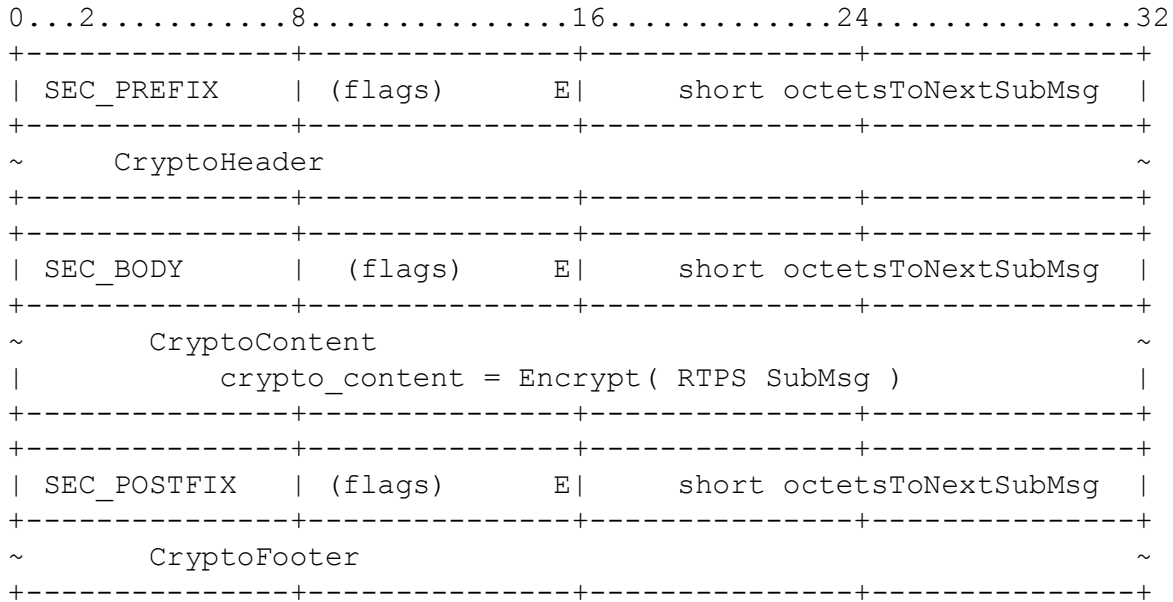
In the above Encrypt indicates the cryptographic transformation performed with AES-GCM using the **SessionKey** and the **InitializationVector** operating on the input RTPS `Submessage`.

The *common_mac* in the **CryptoFooter** is the authentication tag generated by the same AES-GCM transformation where the Additional Authenticated Data is empty.
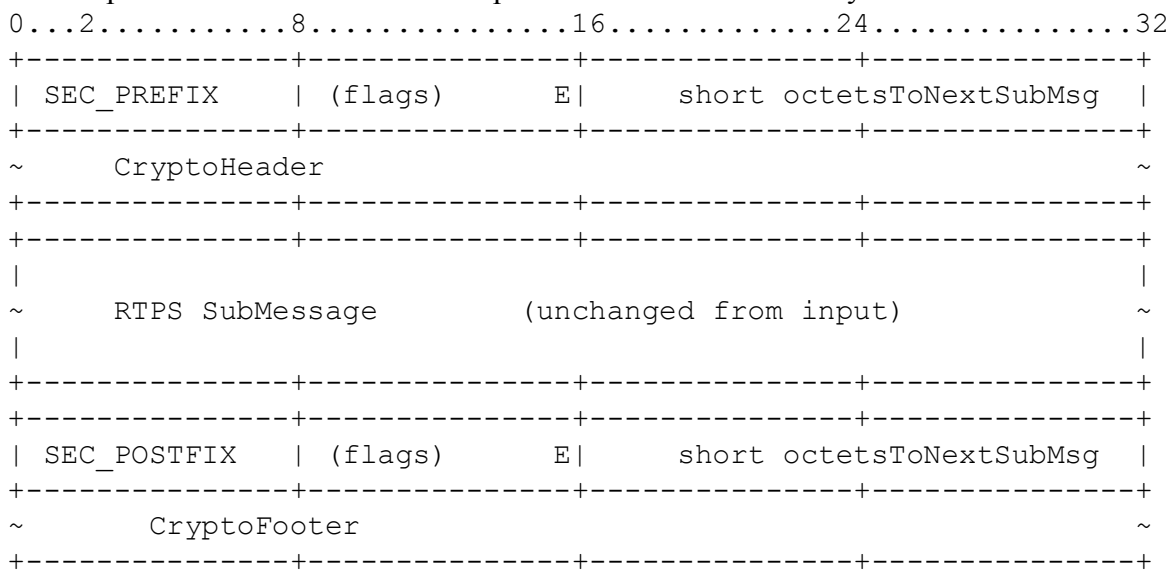
The *receiver_specific_macs* in the CryptoFooter are the AES-GMAC tags computed on the *common_mac* using each of the **SessionReceiverSpecificKey** and the same **InitializationVector**.

9.5.3.3.4.6   Result from encode_rtps_message

The input to this operation is an RTPS message:

```
+--------------+--------------+--------------+--------------+
~        RTPSHdr                                            ~
+--------------+--------------+--------------+--------------+
~        SubMsg1 submessage                                ~
+--------------+--------------+--------------+--------------+
~        SubMsg2 submessage                                ~
+--------------+--------------+--------------+--------------+
|        . . .                                             |
+--------------+--------------+--------------+--------------+
~        SubMsgN submessage                                ~
+--------------+--------------+--------------+--------------+
```

The output in case the transformation performs authentication only shall be:

```
0...2..........8...............16.............24...............32
+--------------+--------------+--------------+--------------+
~        RTPSHdr  (unchanged from input)                   ~
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SRTPS_PREFIX | (flags)     E|     short octetsToNextSubMsg |
+--------------+--------------+--------------+--------------+
~        CryptoHeader                                      ~
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
~        RTPSMessage{ RTPSHdr -> InfoSourceSubMsg }        ~
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SRTPS_POSTFIX |    flags    E|     short octetsToNextSubMsg |
+--------------+--------------+--------------+--------------+
~        CryptoFooter                                      ~
+--------------+--------------+--------------+--------------+
```

The ***common_mac*** in the `CryptoFooter` is the authentication tag generated by the AES-GMAC transformation using the ***SessionKey*** and the ***InitializationVector*** operating on the `RTPSMessage{ RTPSHdr -> InfoSourceSubMsg}`.

`RTPSMessage{ RTPSHdr -> InfoSourceSubMsg}`. Represents the original RTPS Message where the RTPS Header is repaced with an InfoSourceSubMsg with equivalent content.

The ***receiver_specific_macs*** in the `CryptoFooter` are the AES-GMAC tags computed on the ***common_mac*** using each of the ***SessionReceiverSpecificKey*** and the same ***InitializationVector***.

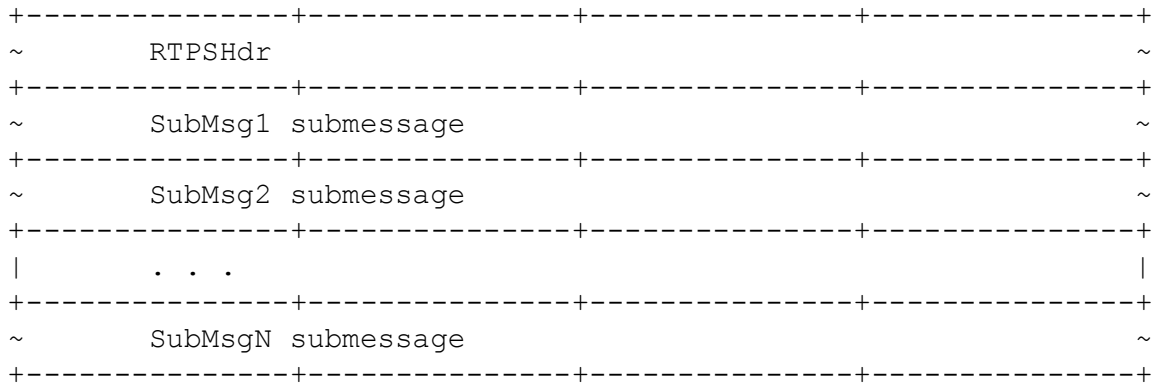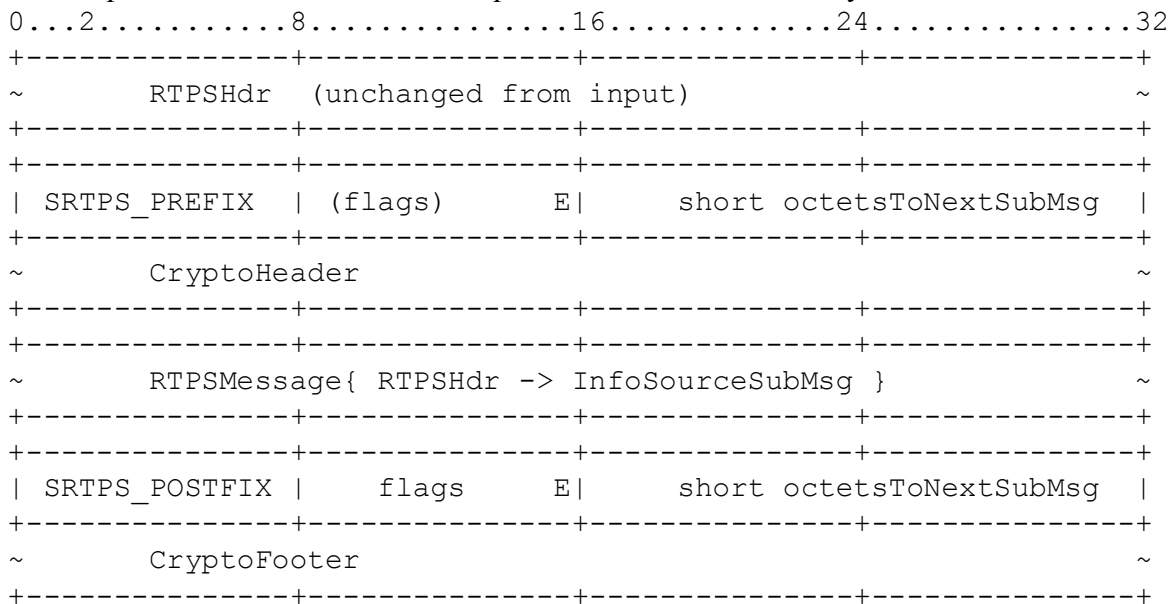The output in case the transformation performs encryption and authentication shall be:

```
+--------------+--------------+--------------+--------------+
~         RTPSHdr  (unchanged from input)                 ~
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SRTPS_PREFIX | (flags)     E|     short octetsToNextSubMsg |
+--------------+--------------+--------------+--------------+
~         CryptoHeader                                     ~
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SEC_BODY     |  (flags)    E|     short octetsToNextSubMsg |
+--------------+--------------+--------------+--------------+
~         CryptoContent                                    ~
|         crypto_content =                                 |
|           Encrypt( RTPSMessage{RTPSHdr -> InfoSourceSubMsg} ) |
+--------------+--------------+--------------+--------------+
+--------------+--------------+--------------+--------------+
| SRTPS_POSTFIX |   flags     E|     short octetsToNextSubMsg |
+--------------+--------------+--------------+--------------+
~         CryptoFooter                                     ~
+--------------+--------------+--------------+--------------+
```

In the above Encrypt indicates the cryptographic transformation performed with AES-GCM using the *SessionKey* and the *InitializationVector* operating on the `RTPSMessage{ RTPSHdr -> InfoSourceSubMsg}`.

The *common_mac* in the *CryptoFooter* is the authentication tag generated by the same CM where the Additional Authenticated Data is empty.

The *receiver_specific_macs* in the CryptoFooter are the AES-GMAC tags computed on the *common_mac* using each of the *SessionReceiverSpecificKey* and the same *InitializationVector*.

9.5.3.3.5   Computation of plaintext from ciphertext

The decrypt operation first checks that the `CryptoTransformIdentifier` attribute in the `CryptoHeader` has the proper `transformation_kind` and also uses the `CryptoTransformIdentifier transformation_key_id` to locate the `MasterKey`, and `MasterSalt`. In case of a re-key the crypto handle (`ParticipantCryptoHandle`, `DatawriterCryptoHandle`, or `DatareaderCryptoHandle`) may be associated with multiple `MasterKeyId` and this parameter allows selection of the correct one. If the `MasterKeyId` is not found associated with the crypto handle the operation shall fail.

The session_id attribute within the `CryptoHeader` is used to obtain the proper `SessionReceiverSpecificKeys` and `SessionKey`. Note that this only requires a re-computation if it has changed from the previously received `SessionId` for that crypto handle.

Given the `InitializationVector` from the `CryptoHeader` and the `SessionKey` the transformation performed to recover the plaintext from the ciphertext is identical to the one performed to go plaintext to ciphertext.

9.5.3.3.6   Computation of the message authentication codes

The message digest is computed on the crypto_header and the ciphertext.

There are two types of message authentication codes (MACs) that may appear.

- The first stored in the ***common_mac*** uses the SessionKey. This MAC may be verified by all the receivers of the message.
- The second type, stored in the ***receiver_specific_macs*** contains MACs that use different `SessionReceiverSpecificKey` whose CryptoTransformIdentifier appears explicitly in the ***receiver_specific_macs***. These MACs use receiver-specific keys that are shared with only one receiver. The key material for these MACs is derived from the `RemoteParticipant2ParticipantKeyMaterial`, the `RemoteWriter2ReaderKeyMaterial`, or the `RemoteReader2WriterKeyMaterial`.

9.6   Builtin Logging Plugin

The builtin Logging Plugin is known as the DDS:Logging:DDS_LogTopic.

The DDS:Logging:DDS_LogTopic implements logging by publishing information to a DDS `Topic` `BuiltinLoggingTopic` defined below.

The `BuiltinLoggingTopic` shall have the Topic name "`DDS:Security:LogTopic`".

The `BuiltinLoggingTopic` shall have the Type `BuiltinLoggingType` defined in the IDL below:

```
enum LoggingLevel {
   EMERGENCY_LEVEL, // System is unusable. Should not continue use.
   ALERT_LEVEL,     // Should be corrected immediately
   CRITICAL_LEVEL,  // A failure in primary application.
   ERROR_LEVEL,     // General error conditions
   WARNING_LEVEL,   // May indicate future error if action not taken.
   NOTICE_LEVEL,    // Unusual, but nor erroneous event or condition.
   INFORMATIONAL_LEVEL, // Normal operational. Requires no action.
   DEBUG_LEVEL
};


@extensibility(FINAL)
struct NameValuePair {
    string name;
    string value;
};



typedef sequence<NameValuePair> NameValuePairSeq;

@extensibility(FINAL)
struct BuiltinLoggingType {
    octet  facility;  // Set to 0x0A (10). Indicates sec/auth msgs
    LoggingLevel severity;
    Time_t timestamp; // Since epoch 1970-01-01 00:00:00 +0000 (UTC)
```

```
    string hostname;   // IP host name of originator
    string hostip;     // IP address of originator
    string appname;    // Identify the device or application
    string procid;     // Process name/ID for syslog system
    string msgid;      // Identify the type of message
    string message;    // Free-form message

    // Note that certain string keys (SD-IDs) are reserved by IANA
    map<string, NameValuePairSeq>  structured_data;
};
```

Knowledge of the `BuiltinLoggingTopic` shall be builtin into the DDS:Auth:PKI-DH `AccessControl` plugin and it shall be treated according to the following topic rule:

```
<topic_rule>
    <topic_expression> DDS:Security:LogTopic</topic_expression>
    <enable_discovery_protection>FALSE</enable_discovery_protection>
    <enable_read_access_control>TRUE</enable_read_access_control>
    <enable_write_access_control>FALSE</enable_write_access_control>
    <metadata_protection_kind>SIGN</metadata_protection_kind>
    <data_protection_kind>ENCRYPT</data_protection_kind>
</topic_rule>
```

The above rule states that any `DomainParticipant` with permission necessary to join the DDS `Domain` shall be allowed to write the `BuiltinLoggingTopic` but in order to read the `BuiltinLoggingTopic` the `DomainParticipant` needs to have a grant for the `BuiltinLoggingTopic` in its permissions document.

9.6.1 DDS:Logging:DDS_LogTopic plugin behavior

The table below describes the actions that the DDS:Logging:DDS_LogTopic plugin performs when each of the plugin operations is invoked.

Table 74 – Actions undertaken by the operations of the builtin Logging plugin

`set_log_options`	Controls the configuration of the plugin. The LogOptions parameter shall be used to take the actions described below:
	If the ***distribute*** parameter is set to TRUE, the DDS:Logging:DDS_LogTopic shall create a `DataWriter` to send the `BuiltinLoggingTopic` if it is FALSE, it shall not.
	The plugin shall open a file with the name indicated in the ***log_file*** parameter.
	The plugin shall remember the value of the ***log_level*** so that it can be used during the `log` operation.

log	This operation shall check if logging was enabled by a prior call to `enable_logging` and if not it shall return without performing any action.
	If logging was enabled, it shall behave as described below:
	The operation shall compare the value of the the ***log_level*** parameter with the value saved during the `set_log_options` operation.
	If the ***log_level*** parameter value is greater than the one saved by the `set_log_options` operation, the operation shall return without performing any action.
	If the ***log_level*** parameter value is less than or equal to the one saved, the `log` operation shall perform two actions:
	• It shall append a string representation of the parameters passed to the log operation to the end of the file opened by the `set_log_options` operation. • If the value of the ***distribute*** option was set on the call to `set_log_options`, the plugin shall fill an object of type `BuiltinLoggingType` with the values passed as arguments to the `log` operation and publish it using the `DataWriter` associated with the `BuiltinLoggingTopic` created by the `set_log_options` operation.
`enable_logging`	This operation shall save the fact that logging was enabled such that the information can be used by the `log` operation.
`set_listener`	This operation shall save a reference to the LoggerListener such that the listener is be notified each time a log message is produced.

10 Plugin Language Bindings

10.1 Introduction

Clause 8 defines the plugin interfaces in a programming-language independent manner using UML. Using the terminology of the DDS specification this UML definition could be considered a Platform Independent Model (PIM) for the plugin interfaces. The mapping to each specific programming languages platform could therefore be considered a Platform Specific Model (PSM) for that programming language.

The mapping of the plugin interfaces to specific programming languages is defined by first defining the interfaces using OMG-IDL version 3.5 with the additional syntax defined in the DDS-XTYPES specification and subsequently applying the IDL to language mapping to the target language.

IDL Types lacking the DDS-XTYPES `@extensibility` annotation shall be interpreted as having the extensibility kind APPENDABLE. This matches the DDS-XTYPES specification implied extensibility of un-annotated types.

For consistency with the DDS specification, the DDS security specification defines language bindings to each of the language PSMs specified for DDS, namely:

- C as derived from the IDL to C mapping
- C++ classic, as derived from the IDL to C++ mapping
- Java classic, as derived from the IDL to Java mapping
- C++ modern, aligned with the DDS-STDC++ specification, this is derived from the IDL to C++11 mapping
- Java modern with the DDS-JAVA5+ specification

10.2 IDL representation of the plugin interfaces

For consistency in the resulting APIs, the mapping from the plugin interfaces defined in clause 8 and the OMG IDL follows the same PIM to PSM mapping rules as the OMG DDS specification (see sub clause 7.2.2 of the DDS specification version 1.2 [1]). A relevant subset of these rules is repeated here. In these rules "PIM" refers to the UML description of the interfaces in clause 8 and PSM refers to the OMG-IDL description of the interfaces that appears in the associated **dds_security.idl** file.

- The PIM to PSM mapping maps the UML interfaces and classes into IDL interfaces. Plain data types are mapped into structures.
- 'Out' parameters in the PIM are conventionally mapped to 'inout' parameters in the PSM in order to minimize the memory allocation performed by the Service and allow for more efficient implementations. The intended meaning is that the caller of such an operation should provide an object to serve as a "container" and that the operation will then "fill in" the state of that objects appropriately.

The resulting IDL representation of the plugin interfaces appears in the file **dds_security.idl** which shall be considered part of the DDS Security specification.

10.3 C language representation of the plugin interfaces

The C language representation of the plugin interfaces shall be obtained applying the IDL to C mapping [5] to the **dds_security.idl** file.

10.4 C++ classic representation of the plugin interfaces

The C++ classic (without the use of the C++ standard library) language representation of the plugin interfaces shall be obtained using the IDL2C++ mapping [7] to the **dds_security.idl** file.

10.5 Java classic

The Java classic language representation of the plugin interfaces shall be obtained using the IDL2Java mapping [6] to the **dds_security.idl** file.

10.6 C++11 representation of the plugin interfaces

This representation is aligned with the DDS-STDC++ PSM.

The C++ classic language representation of the plugin interfaces shall be obtained using the IDL2C++11 mapping [8] to the **dds_security.idl** file with the following exceptions:

1. The IDL module DDS shall be mapped to the C++ namespace **dds** so it matches the namespace used by the DDS-STD-C++ PSM.
2. The mapping shall not use any C++11-only feature of the language or the library (e.g., move constructors, noexcept, override, std::array).
3. Arrays shall map to the `dds::core::array` template defined in the DDS-STD-C++ PSM.
4. The enumerations shall map to the dds::core::safe_enum template defined in the DDS-STD-C++ PSM.
5. The IDL DynamicData native type shall be mapped to the C++ type `dds::code::xtypes::DynamicData` defined in the DDS-STDC++ PSM.

10.7 Java modern aligned with the DDS-JAVA5+ PSM

The Java classic language representation of the plugin interfaces shall be obtained using the IDL2Java mapping [6] to the **dds_security.idl** file with the following exceptions:

1. The IDL module DDS shall be mapped to the Java namespace **org.omg.dds** so it matches the namespace used by the DDS-JAVA5+ PSM.
2. The IDL DynamicData native type shall be mapped to the type `org.omg.dds.type.dynamic.DynamicData` defined in the DDS-JAVA5+ PSM.

Annex A - References

[1] DDS: Data-Distribution Service for Real-Time Systems version 1,2.
http://www.omg.org/spec/DDS/1.2/

[2] DDS-RTPS: Data-Distribution Service Interoperability Wire Protocol version 2.1,
http://www.omg.org/spec/DDS-RTPS/2.1/

[3] DDS-XTYPES: Extensible and Dynamic Topic-Types for DDS version 1.0
http://www.omg.org/spec/DDS-XTypes/

[4] OMG-IDL: Interface Definition Language (IDL) version 3.5 http://www.omg.org/spec/IDL35/

[5] IDL2C: IDL to C Language Mapping, Version 1.0. http://www.omg.org/spec/C/1.0/

[6] IDL2Java: IDL To Java Language Mapping, Version 1.3 http://www.omg.org/spec/I2JAV/1.3/

[7] IDL2C++: IDL to C++ Language Mapping (CPP), Version 1.3
http://www.omg.org/spec/CPP/1.3/PDF

[8] IDL2C++11: IDL To C++11 Language Mapping http://www.omg.org/spec/CPP11/

[9] Transport Layer Security, http://en.wikipedia.org/wiki/Transport_Layer_Security

[10] IPSec, http://en.wikipedia.org/wiki/IPsec

[11] DSA, FIPS PUB 186-4 Digital Signature Standard (DSS).
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[12] Diffie-Hellman (D-H) Key Agreement Method. IETF RFC 2631.
http://tools.ietf.org/html/rfc2631

[13] J. H. Catch *et. al.*, "A Security Analysis of the CLIQUES Protocol Suite",
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.8964

[14] Erramilli, S.; Gadgil, S.; Natarajan, N., "Efficient assignment of multicast groups to publish-subscribe information topics in tactical networks", MILCOM 2008

[15] "RFC 2094 - Group Key Management Protocol (GKMP) Architecture",
http://www.faqs.org/rfcs/rfc2094.html

[16] Raghav Bhaskar, Daniel Augot, Cedric Adjih, Paul Muhlethaler and Saadi Boudjit, "AGDH (Asymmetric Group Diffie Hellman): An Efficient and Dynamic Group Key Agreement Protocol for Ad hoc Networks", Proceedings of New Technologies, Mobility and Security (NTMS) conference, Paris, France, May 2007

[17] Qianhong Wu, Yi Mu, Willy Susilo, Bo Qin and Josep Domingo-Ferrer "Asymmetric Group Key Agreement", EUROCRYPT 2009

[18] "Secure IP Multicast",
http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6552/prod_presentation0900aecd80473105.pdf

[19] Gerardo Pardo-Castellote. "Secure DDS: A Security Model suitable for NetCentric, Publish-Subscribe, and Data Distribution Systems", RTESS, Washington DC, July 2007.
http://www.omg.org/news/meetings/workshops/RT-2007/05-2_Pardo-Castellote-revised.pdf

[20] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman, "The Secure Real-time Transport Protocol (SRTP)" IETF RFC 3711, http://tools.ietf.org/html/rfc3711

[21] Baugher, M., Weis, B., Hardjono, T. and H. Harney, "The Group Domain of Interpretation," IETF RFC 3547, http://tools.ietf.org/html/rfc3547, July 2003.

[22] P. Zimmerman, A. Johnston, and J. Callas, "ZRTP: Media Path Key Agreement for Secure RTP", Internet-Draft, March 2009

[23] F. Andreason, M. Baugher, and D. Wing, "Session description protocol (SDP) security description for media streams," IETF RFC 4568, July 2006

[24] D. Ignjatic, L. Dondeti, F. Audet, P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, November 2006.

[25]   M. Baugher, A. Rueegsegger, and S. Rowles, "GDOI Key Establishment for the STRP Data Security Protocol", http://tools.ietf.org/id/draft-ietf-msec-gdoi-srtp-01.txt, June 2008.

[26]   Bruce Schneier (August 2005). "SHA-1 Broken". Retrieved 2009-01-09. "

[27]   H. Krawczyk, M. Bellare, and R.Canetti, "HMAC: Keyed-Hashing for Message Authentication" IETF RFC 2104, http://tools.ietf.org/html/rfc2104

[28]   Bellare, Mihir (June 2006). "New Proofs for NMAC and HMAC: Security without Collision-Resistance". In Dwork, Cynthia. Advances in Cryptology – Crypto 2006 Proceedings. Lecture Notes in Computer Science 4117. Springer-Verlag.

[29]   S. Turner and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms" IETF RFC 6151, http://tools.ietf.org/html/rfc6151

[30]   Cisco, "Implementing Group Domain of Interpretation in a Dynamic Multipoint VPN", http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6586/ps6660/ps6811/prod_white_paper0900aecd804c363f.html

[31]   CiscoIOS Secure Multicast, http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6552/prod_white_paper0900aecd8047191e.html

[32]   A. Mason. IPSec Overview Part Two: Modes and Transforms. http://www.ciscopress.com/articles/article.asp?p=25477

[33]   R. Canetti, P. Cheng, F. Giraud, D. Pendararkis, J. Rao, P. Rohatgi, and D. Saha, "An IPSec-based Host Architecture for Secure Internet Multicast", Proceedings of the 7th Annual Network and Distributed Systems Security Symposium, San Diego, CA, 2000

[34]   T. Aurisch, and C. Karg, "Using the IPSec architecture for secure multicast communications," 8th International Command and Control Research and Technology Symposium (ICCRTS), Washington D.C., 2003

[35]   J. Zhang and C. Gunter. Application-aware secure multicast for power grid communications, International Journal of Security and Networks, Vol 6, No 1, 2011

[36]   List of reserved RTPS Vendor Ids. http://portals.omg.org/dds/content/page/dds-rtps-vendor-and-product-ids

[37]   PKCS #7: Cryptographic Message Syntax Version 1.5. IETF RFC 2315. http://tools.ietf.org/html/rfc2315

[38]   File expression matching syntax for fnmatch() ; POSIX fnmatch API (IEEE 1003.2-1992 Section B.6)

[39]   X.509 v3. ITU-T Recommendation X.509 (2005) | ISO/IEC 9594-8:2005, Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. http://www.itu.int/itu-t/recommendations/rec.aspx?rec=X.509

[40]   IETF RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, https://tools.ietf.org/html/rfc5280

[41]   ANSI X9.62. ANSI, "Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, 2005

[42]   FIPS 186-4: FIPS Digital Signature Standard (DSS). http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[43]   PKCS#8: Asymmetric Key Packages. IETF RFC 5958. https://tools.ietf.org/html/rfc5958

[44]   PKCS#1: Public-Key Cryptography Standards: RSA Cryptography Specifications Version 2.1 https://tools.ietf.org/html/rfc3447

[45]   [NIST SP 800-38D] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

[46]   [NIST SP 800-90A-R1] NIST Special Publication 800-90A Revision 1. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[47]   IETF RFC 5114 "Additional Diffie-Hellman Groups for Use with IETF Standards" https://tools.ietf.org/html/rfc5114.

[48]   [NIST SP 800-56Ar2] NIST Special Publication 800-56A Revision 2. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

[49]   NIST Suite B Implementer's Guide to NIST SP 800-56A https://www.nsa.gov/ia/_files/SuiteB_Implementer_G-113808.pdf

[50]   IETF RFC 5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF) https://tools.ietf.org/html/rfc5869

[51]   IETF RFC 4514 "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names" https://tools.ietf.org/html/rfc4514

[52]   IETF RFC 2560 "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP" https://tools.ietf.org/html/rfc2560

[53]   IETF RFC 6066 "Transport Layer Security (TLS) Extensions: Extension Definitions" https://tools.ietf.org/html/rfc6066

[54]   IETF RFC 2560 "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension" https://tools.ietf.org/html/rfc6961