Master's Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of measurement

Implementation of actual version of DDSI-RTPS protocol for distributed control in Ethernet network

Jiri Hubacek
Open Informatics

January 2016
Supervisor: Pavel Pisa

Acknowledgement / Declaration

Podekovani..

Prohlasuji..

....

Abstrakt / Abstract

Czech abstract..

Klíčová slova: RTPS, ORTE, Ethernet, Real-Time

Překlad titulu: Implementace aktuální verze protokolu DDSI-RTPS pro distribuované řízení v síti Ethernet

English abstract..

Keywords: RTPS, ORTE, Ethernet, Real-Time

/ Contents

[intro]

Chapter 1 Introduction

The Real-Time Publish-Subscribe (RTPS)[1] is the protocol of Data Distribution Service (DDS)[2] family, supporting Data-Centric Publish-Subscribe in real time and specifying communication in a decentralized network, where multiple nodes needs to send and/or receive data in real time. Specification of protocol is developed by Object Management Group[3] - international, open membership, not-for-profit technology standards consortium, since version 1.0 on February 2002 till version 2.2 on September 2014.

This thesis aims on upgrading ORTE implementation of RTPS protocol to be compatible with the latest standard version 2.2. The structure is as follows. In Chapter 1, there is an introduction to RTPS and ORTE. Chapter 3 compares implemented RTPS 1.0 with the latest RTPS 2.2, chapter 4 covers changes needed for compatibility with version 2.2 of the RTPS protocol and chapter 5 covers testing of new implementation of RTPS protocol in ORTE. In chapter 6, demo application of ORTE called *Shape* for Android is introduced. Original ORTE version based demonstration application has been developed as part of preparation for main work to gain experience with RTPS protocol and its implementation. Security for DDS is discussed in chapter 7.

[over]

Chapter 2
Technology overview

2.1 DDS

There are two main models used in Data Distribution Services. *Centralized* model, where single server for the whole network is needed and all communication goes throw it, introduces single point of failure. When the server is unreachable, the whole network is non-functional. By contrast, *decentralized* approach has no central server, no single point of failure. When one node of the network is non-functional, the rest of the network can continue in data transfers.

2.2 DCPS

In the Data-Centric Publish-Subscribe network, data are sent by *Publishers* and received by *Subscribers*. Node can be *Publisher*, *Subscriber* or both and each node can be interested in different data, timing and reliability. Data-Centric Publish-Subscribe [rfc-1] network is responsible for delivery of right data between right nodes with right parameters.

2.3 RTPS

Real-Time Publish-Subscribe is wire protocol developed to ensure interoperability between DDS implementations. It has been designed to be fault tolerant (decentralized), scalable, tunable, with plug-and-play connectivity and ability of best-effort and reliable communication in real time applications.

2.4 ORTE

Open Real-Time Ethernet (ORTE)[4] is the implementation of RTPS 1.0. It's implemented in Application layer of UDP/IP stack, written in C, under open source license, with own API. Because there are no special requirements, it should be easy to port ORTE to many platforms, where UDP/IP stack is implemented.

2.5 Symbols

AES	■	Advanced Encryption Standard
API	■	Application Programming Interface
CORBA	■	Common Object Request Broker Architecture
CTR	■	Counter
DCPS	■	Data-Centric Publish-Subscribe
DDS	■	Data Distribution Service

DH	■	Diffie-Hellman
DSA	■	Digital Signature Algorithm
HMAC	■	Hash-based Message Authentication Code
IP	■	Internet Protocol
IPsec	■	IP Security
MAC	■	Message Authentication Code
OMG	■	Object Management Group
ORTE	■	Open Real-Time Ethernet
PIM	■	Platform Independent Model
PKI	■	Public Key Infrastructure
PSM	■	Platform Specific Model
RSA	■	Rivest Shamir Adleman
RTPS	■	Real-Time Publish-Subscribe
SEDP	■	Simple Endpoint Discovery Protocol
SHA	■	Secure Hash Algorithm
SPDP	■	Simple Participant Discovery Protocol
SPI	■	Service Plugin Interface
TCP	■	Transmission Control Protocol
TLS	■	Transport Layer Security
UDP	■	User Datagram Protocol
XML	■	EXtensible Markup Language

[compare]

Chapter 3
Actual RTPS protocol

[upgrade]

Chapter 4
Required changes in ORTE

[test]

Chapter 5
Testing of implementation

[shape]

Chapter 6
Shape for Android

6.1 Shape demo

With ORTE implementation of RTPS 1.0 protocol, demo application called Shape is delivered. Shape demo demonstrates the functionality of ORTE - when the color (Blue, Green, Red, Black, Yellow) is choosen, the *Publisher* is created as random shape (Circle, Square, Triangle) moving on the screen. Then, under the topic of color name, object's shape, color and coordinates are published to the network. It's possible to receive and interpret object's data (to see colored shapes moving on the screen) by adding the *Subscribers* of specific topics (colors).

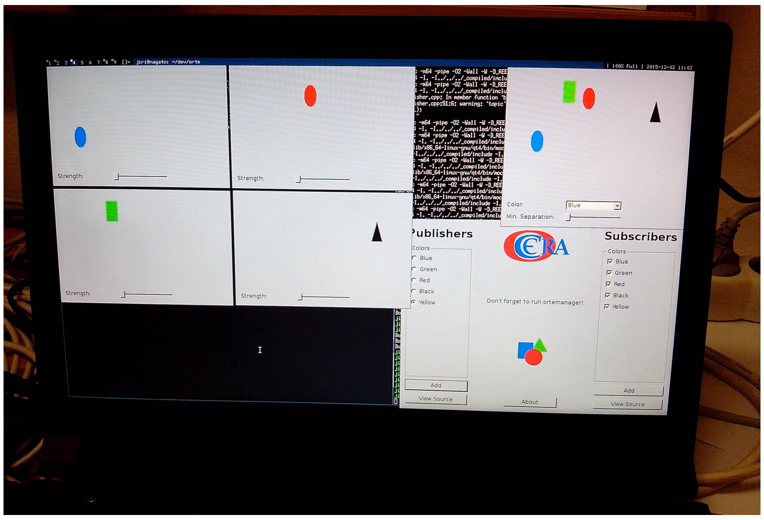


Figure 6.1. Shape demo - *Publishers* and *Subscribers*

6.2 Familiarization with ORTE

The familiarization with ORTE was done by creating demo application for Android compatible with Shape. Because the port of ORTE to Android has been already done in [5] and is available as library, the main task was application design and compatibility ensurance. The application was designed to be as simple as possible. *Publishers* view allows to create new *Publisher* of specific color and random shape, *Subscribers* view allows to set up *Subscribers* of specified colors. Finally, Settings and Help views are present.

6.3 Classes

As in Shape demo, in Shape for Android the *BoxType* class is presented, allowing to create, send and receive objects. *BoxType* consists of *color* (integer), *shape* (integer) and *rectangle* (*BoxRect*), where *BoxRect* is class for storing coordinates - *top_left_x* (short), *top_left_y* (short), *bottom_right_x* (short), *bottom_right_y* (short). The *BoxType* is extension of *MessageData* class delivered with ORTE library for Android. It allows to send and receive objects.

PublisherShape class stores *BoxType* information about *Publisher*, it's properties needed for ORTE, methods for communication with object and prepares data to send. In *Publisher* view, *Publisher* objects are created, stored in *ArrayList* and drawed on screen. Data objects are sent in *Publisher* activity each time objects are redrawn.

SubscriberElement class receives *BoxType* object from ORTE and stores it's data and methods needed for presentation. In *Subscriber* view, all received objects are stored in *ArrayList* and periodically redrawn.

Settings view allows to set up scaling, needed because of various dimensions of screens. It also contains a list of managers - in RTPS 1.0 special application called manager is used for communication of available *Publishers*/*Subscribers* between nodes. In RTPS 2.2 Simple Participant Discovery Protocol (SPDP) and Simple Endpoint Discovery Protocol (SEDP) are used.

Help view contains information about ORTE, Shape and application usage.

6.4 Compatibility

BoxType in Shape and Shape for Android is a little bit different. The reason is just familiarization with ORTE implementation and RTPS protocol, where misunderstooding was not fully avoided. Suggestions for improvements follows.

The first property of *BoxType* is *color*. In Shape demo, *color* is typed as CORBA_octet (macro for uint8_t, 1 byte) and in Shape for Android, *color* is of integer type (4 bytes). The reason why this approach does not break the compatibility is following: each data-type serialized by CORBA is aligned to 4 byte boundary. In this case, object color is first byte and the rest until the boundary is filled by zero bytes. This data representation corresponds to Little Endian in which the message is encoded by default (endianness is operating system dependent), so when Shape for Android deserialize data, Little Endian encoded integer is obtained. It also works in opposite direction - value of the *color* is serialized as integer, encoded as Little Endian and on the side of Shape demo, CORBA_octet is deserialized and 3 zero bytes skipped because of boundary alignment. The problem could arise when *color* would be sent as integer with Big Endian encoding and received as CORBA_octet, because the value of the first byte would be then zero. Also, the problem wouldn't persist in the opposite direction, because endianness is always part of the RTPS message so even node with Big Endian default encoding would receive Little Endian encoded message correctly.

The second property of *BoxType* is *shape*. The type in Shape demo is CORBA_long (macro for int32_t, 4 bytes) and integer (4 bytes) in Shape for Android. Therefore there is no problem with *shape* property.

The last property of *BoxType* is *BoxRect* consisting of coordinates of object. Each value of *BoxRect* is CORBA_short type (2 bytes) in Shape demo and short type (2 bytes) in Shape for Android. Because *BoxRect* is presented as CORBA autonomous data-type, the whole data-type (8 bytes) is aligned to 4 bytes boundary.

The suggestion for the future improvement of Shape demo and Shape for Android is the revision of *BoxType* data-type.

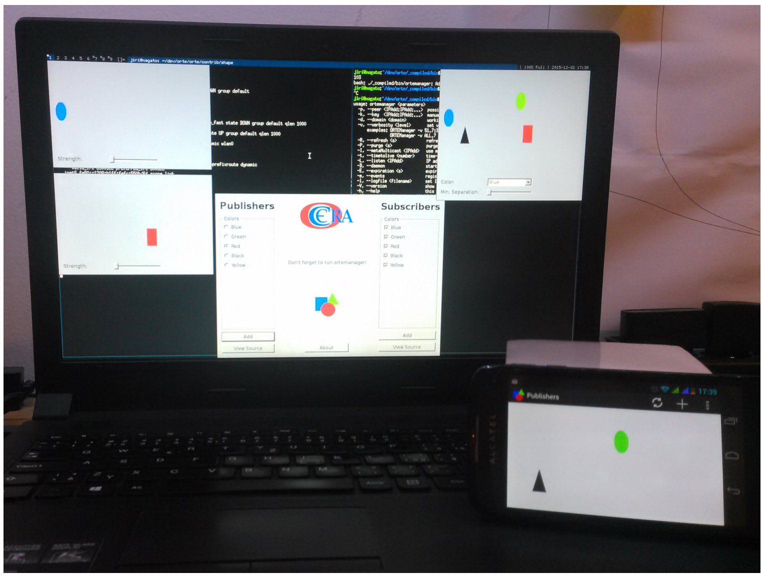


Figure 6.2. Shape for Android - Publishers view

[sec]

Chapter 7 Security for DDS

In the modern world, security is often considered. Technologies for securing communication differs by TCP/IP layers [6] - security at Media access layer consists of preventing deterioration of physical media, environmental noise and access to media. At Network layer, IPsec (IP Security Architecture) protocol is used while Transport layer uses TLS (Transport Layer Security) protocol. In this chapter, Application layer security for DDS standard [7] and possibilities of implementation in RTPS protocol are considered.

7.1 Threats

From point of view of DDS standard, communication takes place in the domain consisting of participants with various number of publishers and subscribers. In this context, Application layer security threats are following:

- Unauthorized[1]) subscription
- Unauthorized publication
- Tampering and replay
- Unauthorized access to data

Unauthorized subscription is a situation when malicious participant receives data for which it is not allowed to. In network infrastructure where access to media is shared, communication runs over multicast or participants sits on one node, it's practically unavoidable to restrict access to data. The solution is making data unreadable for malicious participant - in other words, applying encryption on publisher's side and sharing keys with authenticated subscribers only.

When malicious participant attempts to send data which it is not allowed to, it's called *Unauthorized publication*. For subscriber it's important to receive data only from valid publishers to avoid influence of malicious participant on data. The solution is to include authentication information to data sent by valid publishers so subscribers would be able to recognize data by authenticated publishers from data sent by malicious participant. Two ways how to accomplish authentication of publishers in data are Hash-based message authentication code (HMAC) and digital signature. HMAC creates authentication code using secret key shared between publisher and subscriber. Digital signature is based on private/public key pair - authentication code is created as message digest encrypted by private key of publisher. Each subscriber has access to public key of publisher and can use it to decrypt the authentication code to message digest and compare it with message digest calculated by itself. The point is that these two message digests equals if and only if the authentication code is encrypted by publisher's private key and decrypted by publisher's public key. Digital signature is called *asymmetric*

[1]) Difference between authentication and authorization has to be clear. Authentication is verification of (in this context) participant - that the participant is really the one it claims to be. On the other hand, authorization is process of allowing access to data for already authenticated participant.

cryptography (private/public key pair) and is much slower then *symmetric cryptography* (shared key), therefore the use of HMAC is preferred because of performance reasons.

Valid publisher would send data to subcsriber and malicious participant (in this case, malicious participant will be allowed to subscribe but not to publish). However if the same key is shared between publisher, subscriber and malicious paritcipant, there is no way how to prevent malicious participant to use this shared key for mimicking publisher and sending data to subscriber. This threat is called *Tampering and Replay* and can be solved by sharing different keys between publishers and subscribers. When the communication is taken over multicast, multiple HMACs are needed to be included in data, but this solution is still more powerful than using digital signatures.

In the DDS network, some participants acts as relay participants forwarding data. These participants need to be trusted as valid publishers and subscribers, but it's not always desirable to let them understand data they work with. The solution for *Unauthorized Access to Data* is having different keys for HMAC and data encryption and to share keys for decrypting of data only with desired endpoints.

7.2 Securing of messages

Securing of messages is application dependent - sometimes it's sufficient to encrypt only user-data, in other applications, submessage's metadata as sequence numbers or writer/reader identifiers are needed to be secured too and in the most secure applications, the whole metatraffic submessages are considered confidential. In order to support of different application scenarios, mechanism called *Message Transformation* is introduced. It transforms one RTPS message into another RTPS message so that the original RTPS message or it's submessages may be encrypted into the new one and protected by HMAC.

Because of *Message Transformation*, new submessages and submessage elements are introduced and the questions about interoperability between secured and non-secured implementations of RTPS protocol arises. In implementations of RTPS protocol, unknown submessages should be skipped so the regular user-traffic should not be affected, but there is Discovery also. SPDP is used by DomainParticipants to discover each other, informations as IP address, port, vendor and version are exchanged to bootstrap the communication. Therefore it makes no sense to protect SPDP communication, better to use it for exchange of informations needed to bootstrap the secured system. For both - secured and non-secured implementations of RTPS protocol, *DCPSParticipants* Topic is used in SPDP and there is no new secured Topic for SPDP.

SEDP protocol is used for discovering *publishers* and *subscribers* of each DomainParitcipant. The *DCPSPublications* and *DCPSSubscriptions* Topics are used for communication with non-secured endpoints. However for DomainParticipants supporting DDS Security, *DCPSPublicationsSecure* and *DCPSSubscriptionsSecure* Topics and associated DataWriters (*SEDPbuiltinPublicationsSecureWriter*, *SEDPbuiltinSubscriptionsSecureWriter*) and DataReaders (*SEDPbuiltinPublicationsSecureReader*, *SEDPbuiltinSubscriptionsSecureReader*) are introduced. These Topics should be used for communication that is considered sensitive.

In RTPS protocol, Writer Liveliness Protocol is specified and because data exchange by this protocol could be considered sensitive, DDS Security specifies alternate protected way to exchange liveliness information. *BuiltinParticipantMessageWriter* and *BuiltinParticipantMessageReader* are used to communicate liveliness information with non-secured endpoints. *ParticipantMessageSecure* Topic is introduced with associated *BuiltinParticipantMessageSecureWriter* and *BuiltinParticipantMessageSecureReader*, used to communication liveliness information with endpoints considered sensitive.

Also, there are two completely new builtin Topics:

- ParticipantStatelessMessage
- ParticipantVolatileMessageSecure

ParticipantStatelessMessage Topic is used to perform mutual authentication between DomainParticipants. While the mechanism for participant-to-participant communication already exists, it suffers from weakness of reliable protocol - sequence number prediction. HeartBeat messages containing *first available sequence number* can be abused by malicious participant to prevent other participants to communicate. Therefore new Topic *ParticipantStatelessMessage* with associated *BuiltinParticipantStatelessMessageWriter* (Best-Effort StatelessWriter) and *BuiltinParticipantStatelessMessageReader* (Best-Effort StatelessReader) is introduced.

For key exchange between DomainParticipants, reliable and secure communication is needed. On top of that, DURABILITY Qos needs to be VOLATILE to address only DomainParticipants that are currently in the system. *ParticipantStatelessMessage* is not suitable because it's not reliable nor secured. *ParticipantMessageSecure* Topic is not suitable because it's QoS has DURABILITY kind TRANSIENT_LOCAL rather than VOLATILE (which is required). So new Topic *ParticipantVolatileMessageSecure* with associated *BuiltinParticipantVolatileMessageSecureWriter* and *BuitinParticipantVolatileMessageSecureReader* is introduced.

7.3 Plugin architecture

There are five SPIs:

- Authentication
- Access-Control
- Cryptographic
- Logging
- Data Tagging

Interactions of plugins are shown in figure 7.1.

7.3.1 Authentication plugin

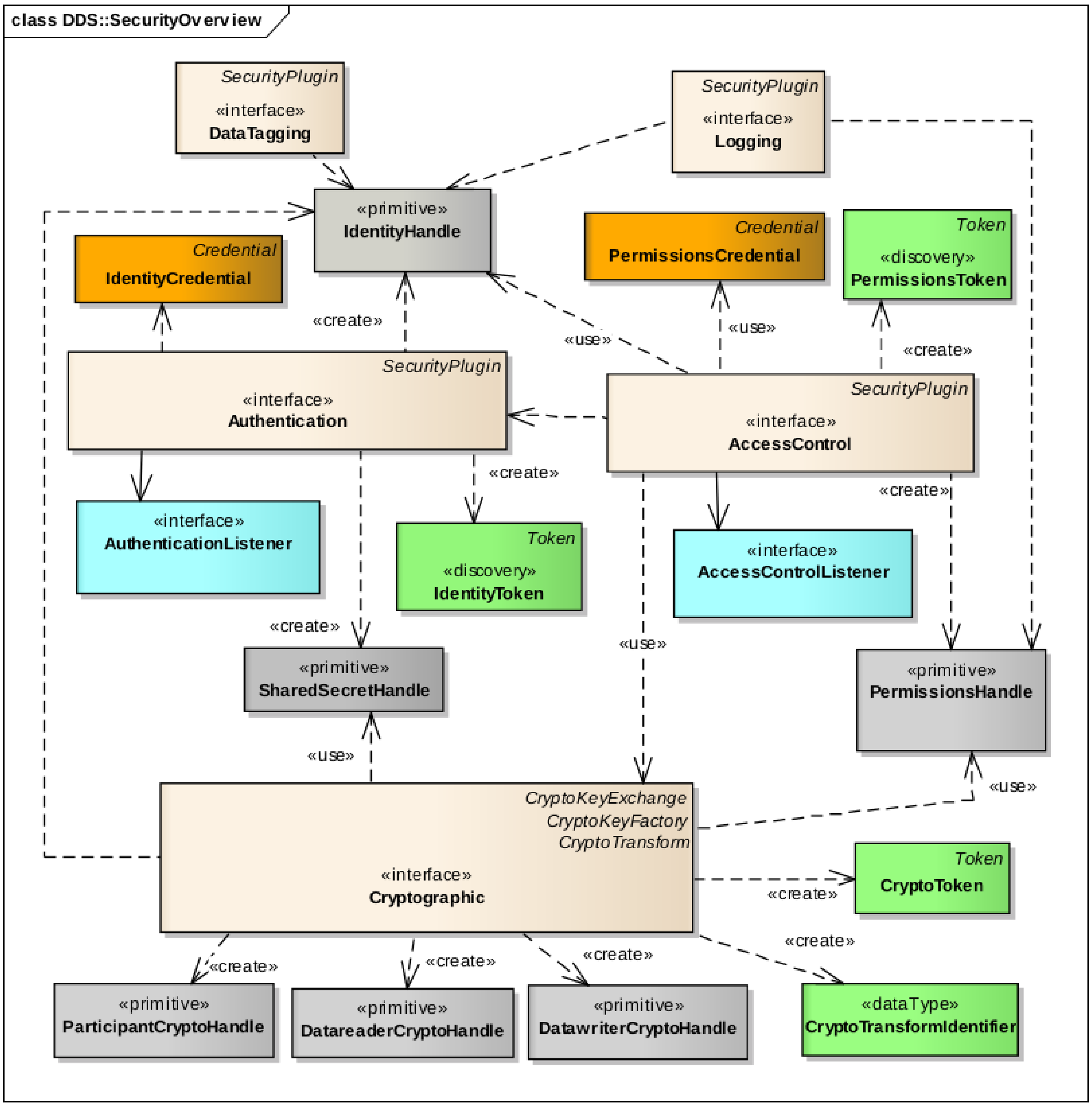
Authentication is process of verifying that (in this case) DomainParticipant is really the one it claims to be. DomainParticipant is authenticated when joining a DDS Domain, mutual authentication is supported and shared secret is established between DomainParitícpants.

7.3.2 Access Control plugin

Ensures authorization - allows or deny protectected operations of DomainParíticipant as join domain, create Topic, publish to Topic or subscribe Topic.

7.3.3 Cryptographic plugin

Encryption, decryption, digests, MAC, HMAC, key generating and exchange, signing and verifying of signatures is ensured by *Cryptographic plugin*. The plugin API has to be general enough to allow specific requirements for cryptographic libraries, encryption and digest algorithms, message authentication and signing users of DDS may need to deploy.


class DDS::SecurityOverview
SecurityPlugin «interface» DataTagging
SecurityPlugin «interface» Logging
«primitive» IdentityHandle
Credential IdentityCredential
Credential PermissionsCredential
Token «discovery» PermissionsToken
SecurityPlugin «interface» Authentication
SecurityPlugin «interface» AccessControl
«interface» AuthenticationListener
Token «discovery» IdentityToken
«interface» AccessControlListener
«primitive» SharedSecretHandle
«primitive» PermissionsHandle
CryptoKeyExchange CryptoKeyFactory CryptoTransform «interface» Cryptographic
Token CryptoToken
«primitive» ParticipantCryptoHandle
«primitive» DatareaderCryptoHandle
«primitive» DatawriterCryptoHandle
«dataType» CryptoTransformIdentifier
«create»
«use»

[pic:sec-pa] **Figure 7.1.** Plugin Architecture Model

7.3.4 Logging plugin

This plugin logs security events of DomainParticipant. Two options of collecting log data are logging all events to a local file and distributing log events securely over DDS.

7.3.5 Data Tagging plugin

Classification of data is performed by *Data Tagging plugin*. It can be used for access control based on tag, message prioritization or even don't have to be used by middleware (RTPS implementation), but by application or service. There are four kinds of tagging:

- Data Writer - used in specification, data received from DataWriter has it's tag.
- Data Instance - each instance of the data has a tag.
- Individual sample - each sample of data instance is tagged individually.
- Per field - the most complex method of tagging.

7.4 Interoperability

Out-of-the-box interoperability of DDS Security implementations is ensured analogously to RTPS implementations - while mandatory *builtin endpoints* ensures that each DomainParticipant is able to discover other DomainParticipants, in DDS Security implementations, each DomainParticipant is able to secure data by at least mandatory *builtin plugins*.

7.4.1 Requirements

This is resume of requirements for builtin plugins by out-of-the-box interoperability as presented in chapter 9.2 of [7]. Following are essential functional requirements for builtin plugins:

- Authentication of DomainParticipants joining a domain
- Access control of applications subscribing to data
- Message integrity and authentication
- Encryption of a data by different keys

Following are functions that should be required by builtin plugins:

- Sending digitally signed data
- Sending data securely over multicast
- Data tagging
- Integrating with open standard security plugins

Following are functions considered useful:

- Access control to certain samples
- Access control to certain attributes within sample
- Permissions for QoS usage by DDS Entities

Non-functional requirements are:

- Performance and Scalability
- Robustness and Availability
- Fit to DDS Data-Centric Information Model
- Reuse of existing security infrastructure and technologies
- Ease of use

7.4.2 Considerations

Usually DDS is deployed in systems where high performance for large number of DomainParticipants is needed, therefore actions performed by plugins shouldn't notably degrade system performance. In practice it means that asymmetric cryptography should be used only for discovery, authentication, session and shared-secret establishment, symmetric cryptography shoud be used for data, use of ciphers, HMACs or digital signatures should be selectable per Topic, there should be possibility of providing integrity via HMAC without data encryption and there should be support for encrypted data over multicast.

DDS used to be deployad in system where *robustness and availability* is considered critical. It's required from system to continue operating even if partial fail occures, so centralized services reprezenting single point of failure have to be avoided, DomainParticipant components have to be self-contained to be able to operate securely, multi-party key agreement protocols should be avoided because of simplicity of discruption and tokens and keys should be compartmentalized as much as possible to avoid situations where multiple applications using same key are compromised if just one of them is compromised.

7.5 Implementation

The implementation of DDS Security into ORTE consists of changes in Modules (presented in chapter 8 of [1]). Introduction of *SecureSubMsg* Submessage and *SecuredPayload* Submessage Element assumes modification of Message Module, Discovery Module is affected by *Builtin Secure Endpoints* - if configured to, discovery of *publishers* and *subscribers* is secured and Behavior Module needs to be modified to include *Builtin Plugins* which ensures security.

7.5.1 Builtin Endpoints

This is the list of *Builtin Endpoints* presented in chapter 7.4.5 of [7]. In order to ensure out-of-the-box compatibility, following *Builtin Secure Endpoints* needs to be implemented:

- SEDPbuiltinPublicationsSecureWriter
- SEDPbuiltinPublicationsSecureReader
- SEDPbuiltinSubscriptionsSecureWriter
- SEDPbuiltinSubscriptionsSecureReader
- BuiltinParticipantMessageSecureWriter
- BuiltinParticipantMessageSecureReader
- BuiltinParticipantVolatileMessageSecureWriter
- BuiltinParticipantVolatileMessageSecureReader

7.5.2 Builtin Plugins

This is resume of *Builtin Plugins* presented in chapter 9.1 of [7]. In order to ensure out-of-the-box compatibility, following *Builtin Plugins* needs to be implemented:

- DDS:Auth:PKI-RSA/DSA-DH (Authentication plugin)
  - Uses PKI with pre-configured shared Certificate Authority
  - RSA or DSA and Diffie-Hellman for authentication and key exchange
- DDS:Access:PKI-Signed-XML-Permissions (Access control plugin)
  - Permissions document signed by shared Certificate Authority
- DDS:Crypto:AES-CTR-HMAC-RSA/DSA-DH (Cryptographic plugin)
  - AES128 for encryption (counter mode)
  - SHA1 and SHA256 for digest
  - HMAC-SHA1 and HMAC-256 for HMAC
- DDS:Tagging:DDS_Discovery (Data Tagging plugin)
  - Send Tags via Endpoint Discovery
- DDS:Logging:DDS_LogTopic (Logging plugin)
  - Logs security events to a dedicated DDS Log Topic

[concl]

Chapter 8
Conclusion

References

[OMG:DDSI-RTPS22][1] Object Management Group (OMG). *The Real-Time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification* . 2014 .
`http://www.omg.org/spec/DDSI-RTPS/2.2/` .

[OMG:DDS][2] Object Management Group (OMG). *Data Distribution Service for Real-Time Systems* . 2007 .
`http://www.omg.org/spec/DDS/` .

[www:OMG][3] *Object Management Group* .
`http://www.omg.org/index.htm` .

[FEE:ORTE][4] *ORTE - Open Real-Time Ethernet* .
`http://orte.sourceforge.net/` .

[FEE:vajnar-bc][5] Martin Vajnar. *ORTE communication middleware for Android OS* . 2014 .

[www:tcp-ip][6] *Core Protocols in the Internet Protocol Suite* .
`http://tools.ietf.org/id/draft-baker-ietf-core-04.html` .

[OMG:DDS-SECURITY][7] Object Management Group (OMG). *DDS Security*. 2014 .
`http://www.omg.org/spec/DDS-SECURITY/` .

Requests for correction

[rfc-1] obrazek