

# IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages

IEEE Vehicular Technology Society

Sponsored by the  
Intelligent Transportation Systems Committee

---

IEEE  
3 Park Avenue  
New York, NY 10016-5997  
USA

IEEE Std 1609.2™-2016  
(Revision of  
IEEE Std 1609.2-2013)



# **IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages**

Sponsor

**Intelligent Transportation Systems Committee  
of the  
IEEE Vehicular Technology Society**

Approved 29 January 2016

**IEEE-SA Standards Board**

**Abstract:** This standard defines secure message formats and processing for use by Wireless Access in Vehicular Environments (WAVE) devices, including methods to secure WAVE management messages and methods to secure application messages. It also describes administrative functions necessary to support the core security functions.

**Keywords:** cryptography, IEEE 1609.2™, security, wireless access in vehicular environments (WAVE)

---

The Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2016 by The Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved. Published 1 March 2016. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-1-5044-0767-0 STD20841  
Print: ISBN 978-1-5044-0768-7 STDPD20841

*IEEE prohibits discrimination, harassment, and bullying.*  
For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.  
No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

## **Important Notices and Disclaimers Concerning IEEE Standards Documents**

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

### **Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents**

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

### **Translations**

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

## **Official statements**

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

## **Comments on standards**

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board  
445 Hoes Lane  
Piscataway, NJ 08854 USA

## **Laws and regulations**

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

## **Copyrights**

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

## **Photocopies**

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

## **Updating of IEEE Standards documents**

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at <http://ieeexplore.ieee.org/xpl/standards.jsp> or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at <http://standards.ieee.org>.

## **Errata**

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for errata periodically.

## **Patents**

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at <http://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

## Participants

At the time this IEEE standard was completed, the Dedicated Short Range Communications Working Group had the following membership:

**Thomas M. Kurihara, Chair**  
**Justin McNew, John Moring, William Whyte, Vice Chairs**

Mike Brown	Carl Kain	Richard Roy
Hanbyeog Cho	John Kenney	Kevin Smith
Hans-Joachim Fischer	Bill Lattin	Jasja Tijink
Ramez Gerges	Jules Madey	Michaela Vanderveen
Aleksandar Gogic	Sean Maschue	George Vlantis
Shubha Gopalakrishna	Jim Misener	Jason Wang
Gloria Gwynne	Frank Perry	Aaron Weinfield
Ronald Hochnadel	Randy Roebuck	

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Nobumitsu Amachi	Piotr Karocki	Alon Regev
Lee Armstrong	John Kenney	Richard Roy
William Byrd	Stuart Kerry	Bartien Sayogo
Keith Chow	Dmitri Khjniak	Kevin Smith
Sourav Dutta	Thomas M. Kurihara	Rene Struik
Richard Edgar	Paul Lambert	Walter Struppler
Marc Emmelmann	Jeremy Landt	Jasja Tijink
Pedro Fernandes	Justin McNew	Steven Tilden
Randall Groves	John Moring	Thomas Tullia
Gloria Gwynne	Michael Newman	Dmitri Varsanofiev
Ronald Hochnadel	Alexandros Nikitas	John Vergis
Werner Hoelzl	Satoshi Oyama	George Vlantis
David Hunter	Venkatesha Prasad	Hung-Yu Wei
Noriyuki Ikeuchi		William Whyte

When the IEEE-SA Standards Board approved this standard on 29 January 2016, it had the following membership:

**Jean-Philippe Faure, Chair**  
**Vacant, Vice Chair**  
**John Kulick, Past Chair**  
**Konstantinos Karachalios, Secretary**

Chuck Adams	Ronald W. Hotchkiss	Gary Robinson
Masayuki Ariyoshi	Michael Janezic	Mehmet Ulema
Ted Burse	Joseph L. Koepfinger*	Yingli Wen
Stephen Dukes	Hung Ling	Howard Wolfman
Jianbin Fan	Kevin Lu	Don Wright
J. Travis Griffith	Annette D. Reilly	Yu Yuan
Gary Hoffman		Daidi Zhong

\*Member Emeritus

## Introduction

This introduction is not part of IEEE Std 1609.2™-2016, IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages.

5.9 GHz Dedicated Short Range Communications for Wireless Access in Vehicular Environments (DSRC/WAVE, hereafter simply WAVE), as specified in a range of standards including those generated by the IEEE P1609 working group, enables vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) wireless communications. This connectivity makes possible a range of applications that rely on communications between road users and road operators, including vehicle safety, public service, commercial fleet management, tolling, and other operations.

With improved communications come increased risks, and the safety-critical nature of many WAVE applications makes it vital that services be specified that can be used to protect messages from attacks such as eavesdropping, spoofing, alteration, and replay. Additionally, the fact that the wireless technology will be deployed in personal vehicles, whose owners have a right to privacy, means that in as much as possible the security services should respect that right and not leak personal, identifying, or linkable information to unauthorized parties.

With this in mind, at the time that IEEE P1609 was established to develop the standards for the WAVE wireless networking protocols, the IEEE also established IEEE P1556 (later renumbered as IEEE Std 1609.2) to develop standards for the security techniques that will be used to protect the services that use these protocols. These applications face unique constraints. Many of them, particularly safety applications, are time-critical: the processing and bandwidth overhead due to security must be kept to a minimum, to improve responsiveness and decrease the likelihood of packet loss. For many applications, the potential audience consists of all vehicles on the road in North America; therefore, the mechanism used to authenticate messages must be as flexible and scalable as possible, and must accommodate the smooth removal of compromised WAVE devices from the system. Additionally, as mentioned above, the privacy of privately owned and operated vehicles, and potentially other personal devices within the WAVE system, must be respected as far as technically and administratively feasible.

## Contents

1. Overview .....	1
1.1 Scope .....	1
1.2 Purpose .....	1
1.3 Document organization.....	2
1.4 Document conventions .....	2
1.5 Testing considerations .....	2
2. Normative references.....	2
3. Definitions, abbreviations, and acronyms .....	4
3.1 Definitions .....	4
3.2 Abbreviations and acronyms .....	10
4. General description.....	12
4.1 WAVE protocol stack overview.....	12
4.2 Secure data service (SDS) .....	15
4.3 Security services management entity (SSME).....	18
4.4 Behavior of SDEEs.....	20
5. Cryptographic operations and validity.....	20
5.1 Certificate validity .....	20
5.2 Signed SPDU validity.....	33
5.3 Cryptographic operations.....	41
6. Data structures .....	46
6.1 Presentation and encoding .....	46
6.2 Integer types .....	46
6.3 Secured protocol data units (SPDUs) .....	47
6.4 Certificates and other security management data structures .....	59
7. Certificate revocation lists (CRLs) and the CRL Verification Entity .....	74
7.1 General .....	74
7.2 CRL Verification Entity specification .....	74
7.3 Data structures .....	75
7.4 CRL: 1609.2 Security envelope.....	80
8. Peer-to-peer certificate distribution (P2PCD).....	85
8.1 General .....	85
8.2 P2PCD operations.....	86
8.3 P2PCD Entity specification .....	96
8.4 Data structures .....	97
9. Service primitives and functions .....	98
9.1 General comments and conventions .....	98
9.2 Identifiers used in the interface specification .....	100
9.3 Sec SAP .....	106
9.4 SSME SAP .....	139
9.5 SSME-Sec SAP .....	162
Annex A (normative) Protocol Implementation Conformance Statement (PICS) proforma .....	167
A.1 Instructions for completing the PICS proforma .....	167

A.2 PICS proforma—IEEE Std 1609.2 .....	169
Annex B (normative) ASN.1 modules.....	179
B.1 1609.2 security services .....	179
B.2 Certificate revocation list (CRL).....	189
B.3 Peer-to-peer certificate distribution (P2PCD) .....	192
Annex C (informative) Specifying the use of IEEE Std 1609.2 by SDEEs.....	194
C.1 General.....	194
C.2 IEEE 1609.2 security profiles .....	194
C.3 IEEE 1609.2 security profile proforma .....	204
C.4 Service Specific Permissions (SSP) .....	206
C.5 Assurance level .....	207
C.6 Recommendations on certificates .....	207
Annex D (informative) Examples and use cases .....	208
D.1 Guidance for SDEE specifiers and implementers .....	208
D.2 Processing CRLs.....	209
D.3 Constructing a certificate chain .....	210
D.4 Peer-to-peer certificate distribution .....	215
D.5 Example data structures .....	222
Annex E (informative) Deployment considerations .....	226
Annex F (informative) Bibliography .....	228

# **IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages**

***IMPORTANT NOTICE: IEEE Standards documents are not intended to ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.***

***This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.***

## **1. Overview**

### **1.1 Scope**

This standard defines secure message formats and processing for use by Wireless Access in Vehicular Environments (WAVE) devices, including methods to secure WAVE management messages and methods to secure application messages. It also describes administrative functions necessary to support the core security functions.

### **1.2 Purpose**

The safety-critical nature of many Wireless Access in Vehicular Environments (WAVE) applications makes it vital that services be specified that can be used to protect messages from attacks such as eavesdropping, spoofing, alteration, and replay. Additionally, the fact that the wireless technology will be deployed in communication devices in personal vehicles as well as other portable devices, whose owners have an expectation of privacy, means that in as much as possible the security services must be designed to respect privacy and not leak personal, identifying, or linkable information to unauthorized parties. This standard describes security services for WAVE management messages and application messages designed to meet these goals.

## 1.3 Document organization

Clause 1 provides an overview of the document. Clause 2 contains the normative references. Clause 3 contains definitions and abbreviations. Clause 4 provides a general description of WAVE Security Services and their use. Clause 5 specifies validity of signed secured protocol data units (SPDUs), correctness of encrypted protocol data units (PDUs), and core cryptographic operations. Clause 6 specifies the encoding and structure of messages generated and consumed by WAVE Security Services. Clause 7 provides a specification of certificate revocation lists, which are a mechanism used to distribute information about certificates that should not be trusted. Clause 8 specifies mechanisms for peer-to-peer certificate distribution. Clause 9 defines the primitives used to communicate between WAVE Security Services and other functional entities.

Annex A provides a Protocol Implementation Conformance Statement (PICS) proforma. Annex B provides ASN.1 modules. Annex C provides a description of the IEEE 1609.2 security profile and a proforma that may be used by developers of applications (or other entities that invoke WAVE Security Services) to specify options for how those applications are to interact with WAVE Security Services. Annex D provides examples, including examples of encoded datagrams and sample process flows, from the point of view of an entity invoking WAVE Security Services rather than from the point of view of WAVE Security Services. Annex E describes other considerations that impact the deployment of a secure communications system using the techniques of this standard. Annex F provides an informative bibliography.

## 1.4 Document conventions

Unless otherwise stated, conventions follow those in IEEE Std 802.11™ [B11]<sup>1</sup>, including conventions for the ordering of information elements within data streams.

Numbers are decimal unless otherwise noted. Numbers preceded by 0x are to be read as hexadecimal, so that 0xFF is equivalent to “FF hexadecimal”. Occasionally, this standard includes representations of octet strings in hexadecimal form; these strings are indicated as hexadecimal on a case-by-case basis.

Figures are used for illustration and are informative, unless otherwise noted.

## 1.5 Testing considerations

The services defined in this standard, with the exception of the peer-to-peer certificate distribution service specified in Clause 8, operate over internal interfaces that are not directly observable in normal operations. Conformance claims made about an implementation of this standard can only be fully tested by direct access to the specific interfaces of that implementation. This standard does not provide a normative interface specification, so an implementation of a test process is not guaranteed a standard interface that may be used to access the implementation under test.

## 2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

---

<sup>1</sup> The numbers in brackets correspond to those of the bibliography in Annex F.

Federal Information Processing Standard (FIPS) 180-4, Secure Hash Standard (SHS), Aug. 2015. Available from <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.<sup>2</sup>

Federal Information Processing Standard (FIPS) 186-4, Digital Signature Standard (DSS), July 2013. Available from <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

Federal Information Processing Standard (FIPS) 197, Advanced Encryption Standard (AES), Nov. 2001. Available from <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

IEEE Std 1363™-2000, IEEE Standard Specifications for Public Key Cryptography.<sup>3, 4</sup>

IEEE Std 1363a™-2004, IEEE Standard Specifications for Public Key Cryptography—Amendment 1: Additional Techniques.

IEEE Std 1609.0™, IEEE Guide for Wireless Access in Vehicular Environments (WAVE)—Architecture.

IEEE Std 1609.3™, Standard for Wireless Access in Vehicular Environments (WAVE)—Networking Services.

IEEE Std 1609.12™, Standard for Wireless Access in Vehicular Environments (WAVE)—Identifier Allocations.

IETF Request for Comments: 3629, UTF-8, A Transformation Format of ISO 10646.<sup>5</sup>

IETF Request for Comments: 5639, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation.

ITU-T Recommendation X.680 (11/2008), Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation, 2008. Available from <http://handle.itu.int/11.1002/1000/9604>.

ITU-T Recommendation X.696 (08/2014), Information Technology—Specification of Octet Encoding Rules (OER), 2014. Available from <http://www.itu.int/rec/T-REC-X.696-201408-I>.

NIMA Technical Report TR8350.2, “Department of Defense World Geodetic System 1984, Its Definition and Relationships with Local Geodetic Systems.” Available from [http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350\\_2.html](http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html)

NIST Special Publication SP 800-38C, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.<sup>6</sup>

Standards for Efficient Cryptography Group, “SEC 1: Elliptic Curve Cryptography,” Version 2.0, May 21, 2009.<sup>7</sup>

Standards for Efficient Cryptography Group, “SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV),” Version 1.0, Jan. 24, 2013.

United Nations Statistics Division, “Composition of Macro Geographical (Continental) Regions, Geographical Sub-Regions, and Selected Economic and Other Groupings,” [referred to as “UN Region Codes”] revision of 31 Oct. 2013. Available from <http://unstats.un.org/unsd/methods/m49/m49regin.htm>.

United States Census, 2010 FIPS Codes for Counties and County Equivalent Entities. Available from <http://www.census.gov/geo/reference/codes/cou.html>.

---

<sup>2</sup> FIPS publications are available from the National Technical Information Service (NTIS), U.S. Dept. of Commerce, 5285 Port Royal Rd., Springfield, VA 22161 (<http://www.ntis.org/>).

<sup>3</sup> IEEE publications are available from The Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA (<http://standards.ieee.org/>).

<sup>4</sup> The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

<sup>5</sup> IETF publications are available from <http://www.ietf.org>.

<sup>6</sup> NIST special publications are available from <http://csrc.nist.gov/publications/nistpubs/>.

<sup>7</sup> SECG publications are available from <http://www.secg.org>.

### 3. Definitions, abbreviations, and acronyms

#### 3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.<sup>8</sup>

**advanced encryption standard (AES):** Federal Information Processing Standard (FIPS) 197, specifying a symmetric block cipher; also, the block cipher specified in that standard.

**application:** A higher layer entity that may make use of WAVE communication facilities.

**application permissions:** The actions a certificate holder is allowed to take as stated in their certificate. Expressed in this standard using Provider Service Identifiers (PSIDs).

**associated certificate** (of a private key): The certificate used to verify signatures generated by that private key.

**associated public key** (of a certificate): The public key that is used to verify signatures associated with a certificate.

**asymmetric cryptographic algorithm:** A cryptographic algorithm that uses two related keys, a public key and a private key, such that the public key is derived from the private key but, given only the public key, it is computationally infeasible to derive the private key.

**authenticated channel:** A logical communications channel such that the receiver has assurance that the sender is who they claim to be, and that modifications to the data can be detected. A channel may be authenticated by applying cryptographic mechanisms to the channel itself, or by checking the transmitted protocol data units (PDUs) by some out-of-band mechanism.

**authentication:** A cryptographic service that provides assurance that the sender of a protocol data unit (PDU) is who they claim to be.

**authorization:** A cryptographic service that provides assurance that the sender of a protocol data unit (PDU) is entitled to certain permissions.

**authorization certificate:** A certificate that is used to validate application protocol data units (PDUs) other than certificate requests.

**block cipher:** A symmetric encryption algorithm that processes data in blocks, typically of 8 or 16 octets.

**certificate:** *See digital certificate.*

**certificate authority (CA) certificate:** A certificate that is used to verify other certificates.

**certificate authority (CA):** An entity that issues certificates to entities that are entitled to them.

**certificate chain:** An ordered set of certificates such that each certificate (except for the top) was issued by the certificate above it in the list. *See also: complete certificate chain; partial certificate chain.*

---

<sup>8</sup>*IEEE Standards Dictionary Online* subscription is available at:  
[http://www.ieee.org/portal/innovate/products/standard/standards\\_dictionary.html](http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html).

**certificate holder:** The entity authorized to use a particular digital certificate to establish trust. The certificate holder can carry out operations using the private key corresponding to the certificate's public key. A certificate holder's certificate is referred to as a *locally held certificate*.

**certificate management information:** Information that allows the secure data service to determine the trustworthiness of certificates and received data.

**certificate revocation list (CRL) distribution center:** An entity that stores and distributes certificate revocation lists (CRLs).

**certificate revocation list (CRL) series (CRL series):** An integer used to assign different certificates issued by the same certificate authority (CA) to distinct sets, such that the certificates in different sets appear on different revocation lists if revoked.

**certificate revocation list (CRL):** A list identifying certificates that have been revoked. *See: revocation.*

**certificate revocation list (CRL) signer:** An entity authorized to sign certificate revocation lists (CRLs).

**certificate signing request (CSR):** A protocol data unit (PDU) sent from an entity to a certificate authority (CA), requesting that the CA issues a certificate on behalf of the entity.

**chains to:** A digital certificate A *chains to* another certificate B if B is above A in the certificate chain (q.v.) from A to the root.

**complete certificate chain:** A certificate chain in which the top certificate is a root certificate and the bottom certificate is an end-entity certificate.

**confidentiality:** A cryptographic service that provides assurance that only the intended recipients of a protocol data unit (PDU) can read it.

**consistency conditions:** Criteria for validity of a signed protocol data unit (PDU) that depend only on the contents of the signed secured protocol data unit (SPDU) and not on the state of the receiver.

**counter mode with cipher block chaining message authentication code (CCM):** A mode of operation of a block cipher where the data is encrypted with a keystream, which in turn is generated by encrypting an incrementing counter, and in turn authenticated with a message authentication code calculated using cipher block chaining mode.

**critical information field:** An information field necessary to establish the validity of a signed secured protocol data unit (SPDU).

**cryptographic type** (of a certificate): How a certificate transfers information about its associated public key (q.v.). Cryptographic types of certificate are: **implicit certificate; explicit certificate.**

**cryptographic verification:** The process of determining whether a signature on a signed secured protocol data unit (SPDU) is consistent with the SPDU and the private key.

**cryptographically secure hash function:** A function that maps an arbitrary-length input into a fixed-length output (the hash value) such that (a) it is computationally infeasible to find an input that maps to a specific hash value and (b) it is computationally infeasible to find two inputs that map to the same hash value. All hash functions used in this document are cryptographically secure hash functions.

**Cryptomaterial Handle:** A reference to a private key and the associated public key or certificate, used to indicate to the secure data service that the referenced key should be used in a particular operation.

**cryptomaterial:** A private key, a public key, or a certificate.

**data plane:** A component of the abstract architecture containing entities that exchange user data.

**decode:** To convert an array of octets into a data structure. *Contrast:* **decrypt, encode.**

**decrypt:** To convert unreadable, encrypted data to readable, decrypted data using a decryption algorithm and a key. *Contrast:* **decode, encrypt.**

**decryption algorithm:** An algorithm that takes as input ciphertext and a key and (if the correct key is provided) produces the original plaintext.

**delta certificate revocation list (delta CRL):** A certificate revocation list that carries information only about certificates that were revoked within a certain time period. *Contrast:* **full certificate revocation list.**

**(permitted) depth of chain** (in a certificate authority (CA) certificate's permissions): The length that a certificate chain is permitted to be, starting from the certificate with which the depth value is associated and ending with an end-entity certificate.

**digital certificate:** A digitally signed document binding a public key to an identity and/or a set of permissions.

**direct hashing:** Creating a hash of particular data by passing that data through a hash function without including any additional data or processing.

**dubious certificate:** A certificate for which the most recent certificate revocation list (CRL) is overdue, i.e., it is scheduled to be issued but has not yet been received.

**elliptic curve cryptography (ECC):** A form of public-key cryptography based on the problem of finding discrete logarithms in a group defined over elliptic curves.

**Elliptic Curve Digital Signature Algorithm (ECDSA):** A digital signature mechanism based on the elliptic curve discrete logarithm problem and standardized in Federal Information Processing Standard (FIPS) 186-4.

**Elliptic Curve Integrated Encryption Scheme (ECIES):** A public-key encryption mechanism based on the elliptic curve discrete logarithm problem.

**encode:** To convert a data structure into an array of octets. *Contrast:* **decode, encrypt.**

**encrypt:** To convert readable data to unreadable, encrypted data using an encryption algorithm and a key. *Contrast:* **decrypt, encode.**

**encryption algorithm:** An algorithm that takes as input plaintext and a key and produces ciphertext.

**encryption certificate:** A certificate that contains an encryption key.

**end-entity certificate:** A certificate used to validate application PDUs or certificate requests.

**enrolment certificate:** A certificate used to validate a certificate request (CSR). *Contrast:* **authorization certificate, certificate authority (CA) certificate.**

**explicit certificate:** A certificate that contains a public key and the certificate authority's signature.

**full certificate revocation list (full CRL):** A certificate revocation list that carries information about certificates that were revoked and have not expired, regardless of when the revocation took place. *Contrast: delta certificate revocation list.*

**global consistency conditions:** Consistency conditions (q.v.) which apply to all protocol data units (PDUs) regardless of the application area in which they are used.

**hash function:** *See: cryptographically secure hash function.*

**hash ID-based revocation:** Revocation (q.v.) which identifies certificates to be revoked via their cryptographic hash.

**hash value:** The output of a hash function.

**IEEE 1609.2 security profile:** A means for specifying options and parameters that are provided to the 1609.2 security services by a particular consumer of those services. May be used as part of the specification of a *secure data exchange entity* (q.v.).

**implicit certificate:** A digital certificate that allows the associated public key to be reconstructed from a reconstruction value and the certificate authority's public key rather than directly providing the associated public key.

**inherited permissions:** Permissions within a subordinate certificate that are communicated by reference to the issuing certificate, rather than stated explicitly within the subordinate certificate.

**integrity:** A cryptographic service that provides assurance that any changes to a protocol data unit (PDU) made after it is validly created will be detected.

**issuing certificate:** The certificate that issues a **subordinate certificate**.

**linkage-based revocation:** Revocation (q.v.) which identifies certificates to be revoked via a linkage value (q.v.) included in the certificate.

**linkage value:** A value included in a certificate that enables that certificate to be revoked via linkage-based revocation (q.v.).

**locally held certificate:** A certificate on a device such that WAVE Security Services on that device may generate a signature on a secured protocol data unit (SPDU) which verifies correctly with that certificate.

**management plane:** A component of the abstract architecture containing functions that manage the entities in the data plane.

**misbehavior:** Behavior that results in devices receiving information that could cause them to take incorrect actions.

**misbehavior authority:** A management component on the network with responsibility for determining which devices are responsible for misbehavior.

**misbehavior reporting:** The activity of providing information to some authority, known as the misbehavior authority, about misbehavior within a particular application area. The subsequent actions taken by the misbehavior authority may be application area specific.

**network byte order:** An ordering of the bytes of an integer such that the byte transmitted first is the byte containing the most significant bit, and the most significant bit is first in that byte.

**non-repudiation (of origin):** A cryptographic service whereby the origin of a message can be demonstrated to a third party, preventing the sender from denying that they produced the message.

**off-cycle certificate revocation list (CRL):** A certificate revocation list (CRL) that is issued before the “next CRL” time indicated in the previous CRL in the CRL series.

**partial certificate chain:** A certificate chain that is not a complete certificate chain, i.e., the top certificate is not a root certificate or the bottom certificate is not an end-entity certificate.

**permitted depth of chain:** See: **depth of chain**.

**peer-to-peer certificate distribution:** A mechanism for allowing devices to learn certificates from peer devices.

**plaintext:** Unencrypted data.

**properly formed certificate:** A certificate that can be parsed correctly according to the data structures and encoding defined in this standard.

**Provider Service Identifier (PSID):** An identifier of an application area. (See IEEE Std 1609.12).

**Provider Service Identifier (PSID) derived from context:** A Provider Service Identifier (PSID) associated with a protocol data unit (PDU) received by a secure data exchange entity (SDEE), where the SDEE does not obtain the PSID from the PDU itself, but by some other means.

**pseudonym certificate:** An authorization certificate (q.v.) that indicates its holder’s permissions but not its holder’s identity.

**pseudonymity:** A property wherein an entity’s permanent or long-lived identities, and its long-term patterns of behavior, cannot be deduced from its network traffic and are only observable by appropriately authorized parties.

**public-key digital signature:** A cryptographically secure checksum that is generated using a private key and verified using a public key.

**reconstruction value:** A value in an implicit certificate that allows the associated public key to be recovered.

**relevance conditions:** Criteria for validity of a signed secured protocol data unit (SPDU) that depend on the local state of the receiver.

**replay attack:** An attack in which an attacker retransmits, possibly with a delay, data that was originally validly transmitted.

**revocation:** The publication by a relevant authority of the information that a particular certificate is no longer to be trusted.

**root certificate:** A self-signed certificate that can be used as a trust anchor to verify other certificates.

**secure data service:** A subset of Wireless Access in Vehicular Environments (WAVE) Security Services providing services that allow secure data service entities to request communications security services to be applied to protocol data units (PDUs).

**secure data exchange entity (SDEE):** An entity that uses IEEE 1609.2 services to secure any communications.

**secure data exchange entity (SDEE)-specific consistency checks:** Consistency checks (q.v.) which are specific to a SDEE (q.v.).

**secure data exchange entity (SDEE)-verified relevance checks:** Relevance checks (q.v.) which cannot be verified within the security services.

**secured protocol data unit (SPDU):** A protocol data unit which has been processed by WAVE Security Services before transmission.

**security envelope:** The additional data added to a protocol data unit by the security services when transforming it into a secured protocol data unit.

**security services management entity (SSME):** The management entity responsible for managing the certificate management information on a device.

**security management message:** A protocol data unit (PDU) used to manage certificates or information about certificates.

**security management:** Operations that support acquiring or establishing the validity of 1609.2 certificates.

**security profile policy:** For an *IEEE 1609.2 security profile* (q.v.), a set of entries in the security profile for which values used by different implementations should be coordinated and for which values may change over time.

**security service-verified relevance conditions:** Relevance conditions (q.v.) that can be tested within the security services.

**self-signed certificate:** A certificate whose signature can be verified with the public key in the certificate.

**Service Specific Permissions (SSP):** A field that indicates the permissions of a particular certificate holder with respect to a particular application area.

**signature:** *See: public key digital signature.*

**subordinate certificate:** The certificate that was issued by an *issuing certificate*.

**symmetric cryptographic algorithm:** A cryptographic algorithm that uses a single key. Knowledge of a symmetric encryption key allows both encryption and decryption. Knowledge of a symmetric authentication key allows generation and verification of message authentication codes.

**Symmetric Cryptomaterial Handle:** A reference to a symmetric key, used to indicate to the secure data service that the referenced key should be used in a particular operation.

**symmetric key:** The key for a *symmetric cryptographic algorithm* (q.v.).

**Transport Layer Security (TLS):** An Internet Engineering Task Force (IETF) protocol, specified in Request for Comments (RFC) 5246 [B13], providing for secure communications over Transmission Control Protocol/Internet Protocol (TCP/IP).

**trigger secure data exchange entity (trigger SDEE):** A secure data exchange entity that sends a secured protocol data unit (SPDU) that triggers peer-to-peer certificate distribution.

**trigger secured protocol data unit (trigger SPDU):** A secured protocol data unit (SPDU) that triggers peer-to-peer certificate distribution.

**trust anchor:** A certificate whose validity does not depend on the validity of other certificates.

**valid certificate:** A certificate that is correctly formed, that has not been revoked or expired, and for which a certificate chain to a trust anchor can be constructed.

**verification type** (of a signed secured protocol data unit (SPDU)): An indication of whether a signed SPDU is to be verified with a public key associated with a certificate, or a public key included in the protocol data unit (PDU) payload. Takes the values *certificate* or *self-signed*.

**Wireless Access in Vehicular Environments (WAVE) device:** A device that is compliant to IEEE Std 1609.3, IEEE Std 1609.4<sup>TM</sup>, and IEEE Std 802.11 communicating outside the context of a basic service set.

**Wireless Access in Vehicular Environments (WAVE) Short Message (WSM):** A packet consisting of a WAVE Short MessageProtocol (WSMP) header and WSM data.

**Wireless Access in Vehicular Environments (WAVE) Short Message Protocol (WSMP):** A protocol specified in IEEE Std 1609.3 that minimizes communications overhead.

### 3.2 Abbreviations and acronyms

AES	Advanced Encryption Standard
APDU	application protocol data unit
ASN.1	Abstract Syntax Notation 1
CA	certificate authority
CCM	counter mode with cipher block chaining message authentication code
CMH	Cryptomaterial Handle
COER	Canonical Octet Encoding Rules
CRACA	Certificate Revocation Authorizing Certificate Authority
CRL	certificate revocation list
CSR	certificate signing request
DSRC	dedicated short range communications
ECC	elliptic curve cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EDCA	enhanced distributed channel access
FIPS	Federal Information Processing Standard
GNSS	Global Navigation Satellite System

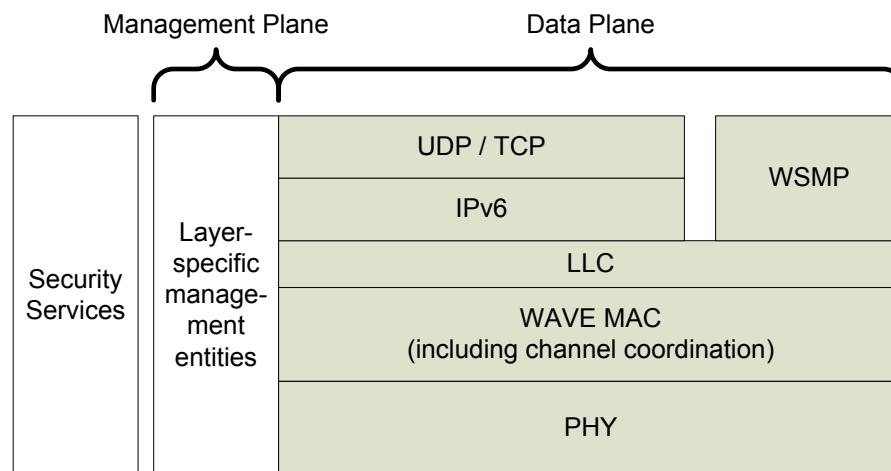
GPS	Global Positioning System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv6	Internet Protocol version 6
ITS	intelligent transportation systems
MAC	medium access control, or message authentication code
NIST	National Institute for Standards and Technology
OBU	on-board unit
P2PCD	peer-to-peer certificate distribution
PDU	protocol data unit
PHY	physical layer
PSID	Provider Service Identifier
RSU	roadside unit
RFC	Request for Comments
SAE	Society of Automotive Engineers
SAP	Service Access Point
SCMH	Symmetric Cryptomaterial Handle
SDEE	secure data exchange entity
SDS	secure data service
SSME	security services management entity
SP	special publication
SPDU	secured protocol data unit
SSP	Service Specific Permissions
TLS	Transport Layer Security
TAI	International Atomic Time
TTP	trusted third party
UTC	Coordinated Universal Time
UTF	Unicode Transformation Format

V2I	vehicle-to-infrastructure
V2V	vehicle-to-vehicle
WAVE	Wireless Access in Vehicular Environments
WGS	World Geodetic System
WME	WAVE Management Entity
WSM	WAVE Short Message
WSMP	WAVE Short Message Protocol

## 4. General description

### 4.1 WAVE protocol stack overview

Wireless Access in Vehicular Environments (WAVE) provides a communication protocol stack optimized for the vehicular environment, employing both customized and general-purpose elements as shown in Figure 1. WAVE supports both IP- and non-IP-based data transfers, although individual devices might support only one networking stack. Non-IP-based data transfers are supported through the WAVE Short Message Protocol (WSMP) specified in IEEE Std 1609.3. Channel coordination is a collection of extensions to the IEEE 802.11 medium access control (MAC) specified in IEEE Std 1609.4 [B12]. The WAVE Management Entity (WME) and corresponding network services are specified in IEEE Std 1609.3. WAVE Security Services are specified in this standard. IEEE Std 1609.0 provides a description of the WAVE system architecture and operations.



**Figure 1—WAVE reference model**

This standard specifies a collection of WAVE Security Services available to processes running on platforms including, but not limited to, WAVE devices. The WAVE Security Services are shown in Figure 2 and consist of WAVE Internal Security Services and WAVE Higher Layer Security Services.

WAVE Internal Security Services are:

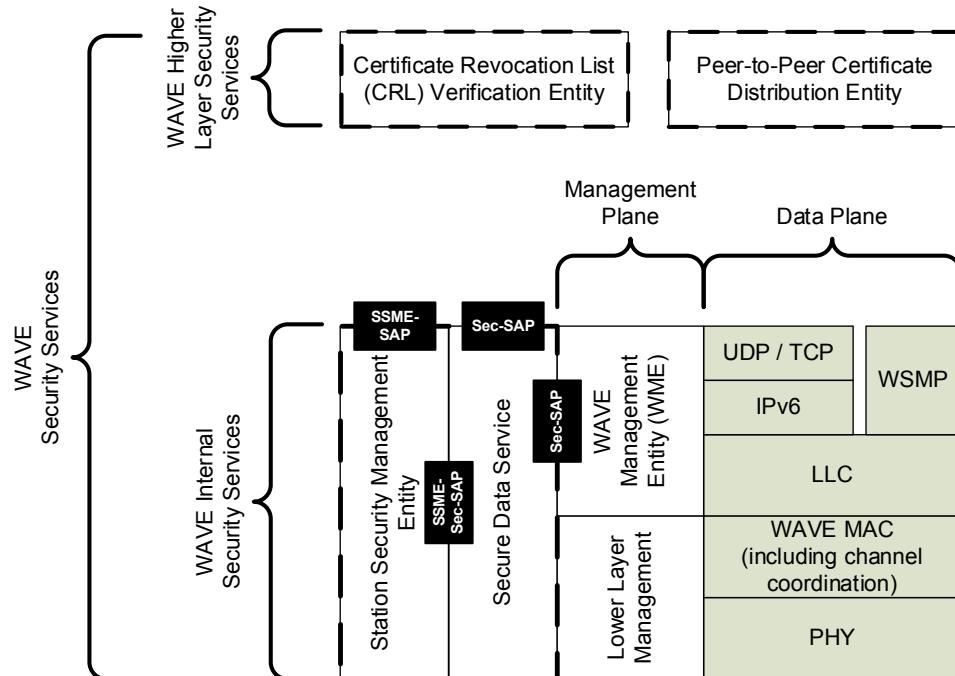
- *Secure data service (SDS)*: Transforming unsecured protocol data units (PDUs) into secured protocol data units (SPDUs) to be transferred between entities, and processing SPDUs on reception, including transforming SPDUs into unsecured PDUs. The additional data added to a PDU when it is transformed into a SPDU is referred to as the *security envelope*. An entity that uses the secure data service is referred to as a *secure data exchange entity (SDEE)*.
- *Security management*: Managing information about certificates as specified in 4.3.

WAVE Higher Layer Security Services are:

- *Certificate revocation list (CRL) verification entity (CRLVE)*: Validates incoming CRLs and passes the related revocation information to the SSME for storage as specified in 5.1.3 and Clause 7.
- *Peer-to-peer certificate distribution (P2PCD) entity (P2PCDE)*: Enables peer-to-peer certificate distribution as specified in Clause 8.

The services and entities within the WAVE Security Services are shown in Figure 2, which also shows Service Access Points (SAPs) that support communications between WAVE Security Services entities and other entities. This standard specifies information flows to support security processing via primitives defined at these SAPs. The Sec-SAP is used by higher layer entities and by the WME. Additionally, the certain SDS operations involve invoking certain primitives across the SSME-SAP as specified in 4.3.

Information elements passed across the SAPs in this standard are assumed to be secure and trustworthy. This standard does not provide mechanisms to ensure the trustworthiness of these information elements.

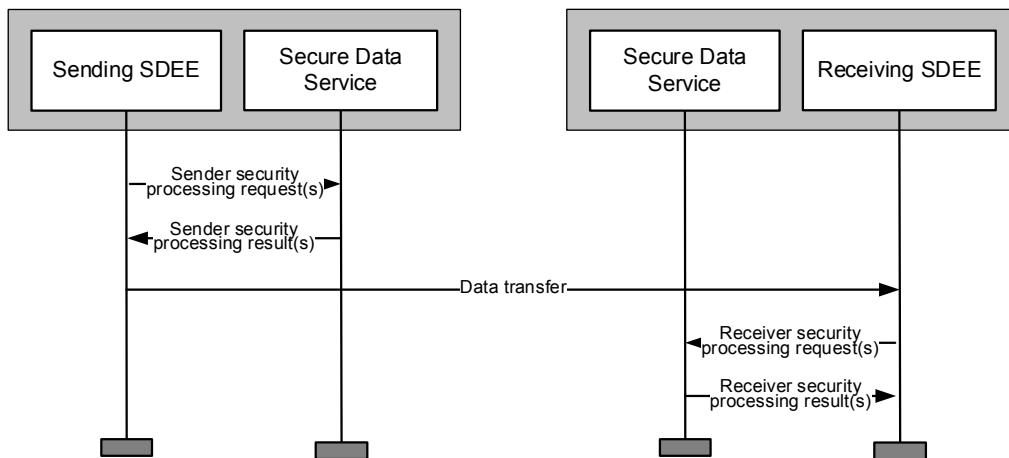


**Figure 2—WAVE protocol stack showing detail of WAVE Security Services**

Figure 3 shows the general model for security processing using the SDS. The SDS is invoked by a secure data exchange entity (SDEE) with a request to process data; the resulting processed data is returned to the invoking SDEE. A secure data exchange involves two SDEEs, one sending and one receiving. Transmission and reception of the SPDU are not specified in this standard.

The sending SDEE invokes the secure data service to perform sender-side security processing. The result of the processing is a SPDU which is returned to the sending entity. The sending SDEE invokes the secure data service at least once, and possibly multiple times, prior to transmission of a SPDU.

The receiving SDEE invokes the secure data service to perform security processing on the contents of a received SPDU. The results of the processing, which may include a SPDU and may include additional information about the SPDU, are returned to the receiving SDEE. Complete processing of a received SPDU might require multiple invocations of the SDS.



**Figure 3—Process flow for use of 1609.2 secure data service**

At a minimum, an implementation of WAVE Security Services shall support at least one of the following:

- Generate signed SPDU (see 4.2.2.3)
- Verify signed SPDU (see 4.2.2.3.2)
- Generate encrypted SPDU (see 4.2.2.4)
- Decrypt encrypted SPDU (see 4.2.2.3.3)
- The CRL Verification Entity (see Clause 7)
- The Peer-to-Peer Certificate Distribution Entity (P2PCDE) (see 8.3)

## 4.2 Secure data service (SDS)

### 4.2.1 Secured protocol data units (SPDUs)

The SDS operations create or process SPDUs. SPDUs are the datagrams that are exchanged between instances of SDEEs that make use of the security services. SPDUs are used for two purposes:

- a) To provide cryptographic protection of the contents
- b) To provide security management information to be exchanged to allow the correct processing of other SPDUs

SPDUs may be of type *unsecured*, *signed*, or *encrypted*. A SPDU may contain another SPDU of the same or different type.

### 4.2.2 Secure data service

#### 4.2.2.1 SDEE identifier

If an implementation of the secure data exchange services supports being invoked by multiple SDEEs, the SDS distinguishes between different SDEEs that invoke them for purposes of replay detection when verifying signed messages (see 5.2.4.2.6), and peer-to-peer certificate distribution (see Clause 8). The mechanism by which this is supported is implementation specific. In the primitives defined in Clause 9, SDEEs are distinguished by a SDEE identifier value which is distinct for distinct SDEEs.

#### 4.2.2.2 Generate SPDUs

##### 4.2.2.2.1 Types of SPDUs

This standard defines the following types of SPDUs: *unsecured*, *signed*, and *encrypted*.

##### 4.2.2.2.2 Unsecured SPDUs

An unsecured SPDU is an encoded IEEE1609Dot2Data indicating content of type `unsecuredData` as defined in Clause 6. The SDS interface specified in this standard does not provide a primitive to create an unsecured SPDU as this can be implemented trivially.

##### 4.2.2.2.3 Signed SPDUs

The SDS may provide the service of generating a signed SPDU. This service provides:

- a) *Authenticity*—assurance that the sender is who they claim to be
- b) *Authorization*—assurance that the sender is entitled to the privileges they request
- c) *Integrity*—assurance that any changes to the SPDU after it is signed can be detected
- d) *Non-repudiation (of origin)*—the ability to demonstrate authenticity, authorization, and integrity to a third party

The SDS is requested to generate a signed SPDU via Sec-SignedData.request. The SDS returns the result of the request to the requesting SDEE via Sec-SignedData.confirm. The result shall be one of the following:

- On success, an octet string containing an encoded Ieee1609Dot2Data as defined in Clause 6 containing a signed SPDU that is valid by the criteria specified in 5.2
- On failure, because it was not possible for the SDS to generate a valid signed SPDU, an indication of the reason for failure

When generating a signed SPDU, the SDS interacts with the SSME to carry out activities to support peer-to-peer certificate distribution (P2PCD) as defined in Clause 8, if the invoking SDEE so requests.

#### **4.2.2.2.4 Encrypted SPDUs**

The SDS may provide the service of generating an encrypted SPDU. This provides:

- *Confidentiality*—assurance that only the intended recipient(s) can read the contents of the SPDU

The SDS is requested to encrypt data via Sec-EncryptedData.request. The caller provides the SDS with a single input PDU and one or more *encryption keys*, which may be public or symmetric. The PDU is encrypted using the encryption keys in such a way that the holder or holders of a decryption key corresponding to any of the encryption keys can decrypt the PDU as specified in 5.3.4. Each of the encryption keys used is referred to as a *recipient key*, and the holder of the corresponding decryption key is referred to as a *recipient*.

The SDS returns the result of the request to the requesting SDEE via Sec-EncryptedData.confirm. The result shall be one of the following:

- On success, an octet string containing an encoding of an Ieee1609Dot2Data containing an encrypted PDU that is valid by the criteria specified in 5.3.4
- On failure, an indication of the reason for failure

#### **4.2.2.2.5 SPDU with multiple layers of cryptographic protection**

To create a SPDU with multiple layers of cryptographic protection (signed and then encrypted, for example), a SDEE invokes the SDS multiple times, passing the output of one invocation of the SDS as the payload to the next invocation. The SPDU produced by each step contains the SPDU from the previous step (or the original PDU) as the payload and can be processed by the security services (if a SPDU) or by a peer SDEE (if the original PDU).

#### **4.2.2.3 Processing received SPDUs**

##### **4.2.2.3.1 Preprocessing**

The SDS may provide the service of preprocessing a received SPDU. This enables peer SDSs to exchange management information and to extract information that can be used by the invoking SDEE to decide whether to process the SPDU further.

The SDS shall provide this service if it supports peer-to-peer certificate distribution (P2PCD) as defined in Clause 8.

The SDS shall provide this service if it allows a signer identifier to be of type digest as defined in 6.3.24.

Otherwise, provision of this service is optional.

The SDS is requested to preprocess a received SPDUs via Sec-SecureDataPreprocessing.request. The SDS returns the result of the request to the requesting SDEE via Sec-SecureDataPreprocessing.confirm. The result is:

- The type of the SPDUs
- If the SPDUs was of type signed:
  - The Service Specific Permissions of the signer as defined in 5.2.3.3.3
  - The assurance level of the signer's certificate as defined in 6.4.27
  - The Earliest Next CRL Time of any certificate in the chain as defined in 5.1.3.6

When providing this service, the SDS extracts security management information and passes it to the SSME to support P2PCD and digest-form SignerIdentifier structures.

#### **4.2.2.3.2 Verifying signed SPDUs**

The SDS may provide the service of verifying a signed SPDUs.

The SDS is requested to verify a signed SPDUs via Sec-SignedDataVerification.request. The SDS returns the result of the request to the requesting SDEE via Sec-SignedDataVerification.confirm. The result shall be a correct indication of whether a signed SPDUs is valid, meaning that it meets the three validity conditions specified in 5.2.1.

#### **4.2.2.3.3 Decrypting encrypted SPDUs**

The SDS may provide the service of decrypting an encrypted SPDUs.

The SDS is requested to decrypt an encrypted SPDUs via Sec-EncryptedDataDecryption.request. The SDS returns the result of the request to the requesting SDEE via Sec-EncryptedDataDecryption.confirm. The result shall be:

- On success, a SPDUs that is the correct decryption of the ciphertext within the encrypted SPDUs.
- On failure, an indication of the reason for failure. A decryption attempt might fail because the data was not validly encrypted, because valid encrypted data was not received correctly, or because the relevant decryption key is not known to the SDEE.

### **4.2.3 Cryptomaterial**

Cryptographic operations for SDS make use of data of the following types:

- Symmetric keys.
- Private keys and associated certificates. The *associated certificate* for a private key is the certificate for which the associated public key verifies signatures generated with that private key and which was valid at the time of signing with the private key.
- Digital certificates held by peer entities, for which no private key is stored by the SDS.

Symmetric keys, and private keys and associated certificates, are referred to as *cryptomaterial* in this standard. Cryptomaterial is used by the SDS in the following operations:

- Generate signed SPDUs (private keys and certificates only)
- Decrypt encrypted PDUs (private keys or symmetric keys)

The interfaces to the SDS in this standard do not pass secret or private keys, but represent them via the Cryptomaterial Handle (CMH). The CMH is an abstraction of private/secret key storage used in the definition of the interfaces and is defined in 9.2.2.

#### **4.2.4 Peer-to-peer certificate distribution**

The SDS may also provide functionality in support of peer-to-peer certificate distribution operations. A SDS implementation may support the responder role functionality specified in 8.2.4.2. A SDS implementation may support the requester role functionality specified in 8.2.4.1. A SDS implementation that supports requester role functionality shall also support responder role functionality.

#### **4.2.5 1609.2 security profile**

The information elements used by the SDS operations are specified in 9.2.2 as (sometimes optional) parameters to primitives; the SPDU data structures and their encodings are specified in Clause 6. The *IEEE 1609.2 security profile* specified in Annex C is a format suggested for use by the specifier of a SDEE as a compact way to specify which SDS parameters are used and which values they should take for that particular SDEE.

Examples of security profiles can be found in Clause 7 of this document (for use with CRLs) and in IEEE Std 1609.3 (for use with WAVE Service Advertisements).

### **4.3 Security services management entity (SSME)**

#### **4.3.1 General**

The SSME stores certificates and information about certificates. The information stored by the SSME relates to both certificates for which the corresponding private key is stored by the SDS (i.e., locally held certificates) and certificates for which the corresponding private key is not stored by the SDS, (e.g., those belonging to peer SDEEs and to Certificate Authorities [CAs]).

The SSME stores the following information relating to each certificate that it manages:

- The certificate data.
- The last time relevant revocation information was received, if any (see 5.1.3).
- The next time revocation information is expected to be received, if any (see 5.1.3). This time may be in the past.
- The certificate's verification status, which is one of the following:
  - Verified and trusted, meaning that it satisfies all the validity conditions of 5.1.
  - Chain does not end in trust anchor (see 5.1.2.1).

- Chain too long for implementation (see 5.1.2.3).
- Not cryptographically valid (see 5.1.2.3).
- Not yet cryptographically validated.
- Inconsistent permissions in chain (see 5.1.2.4).
- Revoked (see 5.1.3).
- Dubious (see 5.1.3.6).
- Certificate or chain contains unsupported critical information fields (see 5.2.5).
- Invalid encoding (certificate or certificate in its chain is not a valid encoding of the data structures in Clause 6).
- Whether or not the certificate is a trust anchor.

This standard refers to certificates whose management information is available within the SSME as certificates that are “known to” or “managed by” the SSME. When certificate information is added to an instance of a SSME, the SSME makes it available to all SDEEs and instances of the SDS that have access to that SSME.

As illustrated in Figure 2, the SSME has two Service Access Points (SAPs) through which other entities communicate with it to obtain and update security management information: one for use primarily by the SDS (SSME-Sec-SAP), and one for use by the SDS and by other entities (SSME-SAP).

The SSME-SAP is used:

- To add information about certificates via SSME-AddCertificate.request and SSME-AddCertificate.confirm
- To provide information about managed certificates via SSME-CertificateInfo.request and SSME-CertificateInfo.confirm
- To request that a certificate is verified via SSME-VerifyCertificate.request and SSME-VerifyCertificate.confirm
- To request deletion of information about a certificate via SSME-DeleteCertificate.request and SSME-DeleteCertificate.confirm
- To add a certificate to its list of trust anchors via SSME-AddTrustAnchor.request and SSME-AddTrustAnchor.confirm
- To update revocation information for a certificate via SSME-AddHashIdBasedRevocation.request, SSME-AddHashIdBasedRevocation.confirm, SSME-AddIndividualLinkageBasedRevocation.request, SSME-AddIndividualLinkageBasedRevocation.confirm, SSME-AddGroupLinkageBasedRevocation.request, and SSME-AddGroupLinkageBasedRevocation.confirm
- To update information about CRLs relevant to managed certificates via SSME-AddRevocation-Info.request and SSME-AddRevocationInfo.confirm
- To provide information about CRLs relevant to managed certificates via SSME-RevocationInformationStatus.request and SSME-RevocationInformationStatus.confirm
- To enable application processes associated with peer-to-peer certificate distribution via SSME-P2pcdResponseGenerationService.request, SSME-P2pcdResponseGenerationService.confirm, and SSME-P2pcdResponseGeneration.indication

The SSME-Sec-SAP is used:

- To provide information about replayed PDUs via SSME-Sec-ReplayDetection.request and SSME-Sec-ReplayDetection.confirm
- To provide information to enable peer-to-peer certificate distribution as specified in Clause 8 via SSME-Sec-IncomingP2pcdInfo.request, SSME-Sec-IncomingP2pcdInfo.confirm, SSME-Sec-OutgoingP2pcdInfo.request, and SSME-Sec-OutgoingP2pcdInfo.confirm
- To enable configuration of peer-to-peer certificate distribution via SSME-P2pcdConfiguration.request and SSME-P2pcdConfiguration.confirm

In addition to the SSME-Sec-SAP primitives, the SDS operations in this standard invoke the following SSME-SAP primitives:

- SSME-AddCertificate.request
- SSME-CertificateInfo.request
- SSME-VerifyCertificate.request
- SSME-RevocationInformationStatus.request

The SSME-SAP specification in this document assumes that all information provided to the SSME is trustworthy; how this is ensured is outside the scope of this standard.

Any certificate information added to an instance of a SSME is made available to all SDEEs and functional entities that have access to that SSME.

#### **4.3.2 Peer-to-peer certificate distribution**

The SSME may also support peer-to-peer certificate distribution operations as specified in Clause 8. A SSME implementation may support the responder role functionality specified in 8.2.4.2. A SSME implementation that supports requester role functionality shall also support responder role functionality.

### **4.4 Behavior of SDEEs**

This standard specifies WAVE Security Services and does not specify SDEE behavior. However, WAVE Security Services protect SDEEs most effectively if used appropriately. D.1 provides guidance for implementers of SDEEs.

## **5. Cryptographic operations and validity**

### **5.1 Certificate validity**

#### **5.1.1 Certificate contents**

A certificate is a data structure used to transport the following information:

- A public key used to verify digital signatures (the “verification key”)

- The permissions associated with that public key
- Optionally, a public key that may be used to encrypt data
- An identifier for the issuer
- Information that may be used to determine whether or not the certificate has been revoked as specified in 5.1.3
- A cryptographic demonstration that the issuer authorized the linkage between the public key and the permissions

The entity that uses the private key corresponding to the public key is referred to as the *certificate holder*.

“Permissions” consist of:

- Geographic permissions: the region within which the certificate is valid, if relevant
- Validity period: the time period within which the certificate is valid
- Application permissions: the activities other than certificate request and issuance that the holder is allowed to perform
- Certificate issuance permissions: The type(s) of certificate, if any, that the holder is permitted to issue
- Certificate request permissions: The type(s) of certificate request, if any, that the holder is permitted to generate

Provider Service Identifiers (PSIDs) are used in certificates to specify permitted application areas. The PSID is defined in IEEE Std 1609.12™. A Service Specific Permission (SSP) is provided (explicitly or implicitly) for each PSID in the Application Permissions, identifying specific sender permissions within that PSID’s application area. The syntax and semantics of the SSP are specific to each PSID value.

When a certificate is being used to authorize application PDUs it is referred to as an *authorization certificate*.

Certificate issuance permissions, i.e., the permissions that govern what certificates a CA is authorized to issue, are expressed using the following information elements (see 6.4.30 for a full specification):

- One or more PSIDs.
- For each PSID, the SSP Range, which identifies the SSPs associated with that PSID for which the CA is permitted to grant permissions.
- The permissible length(s) of the certificate chain from the certificate containing these issuance permissions to the certificate that signs the PDU. This length is referred to as the permitted *depth*. Certificate chain length is defined in 5.1.2.1.
- The end-entity type permissions, which indicate whether the ultimate end-entity certificate permits application operations, certificate request operations, or both.

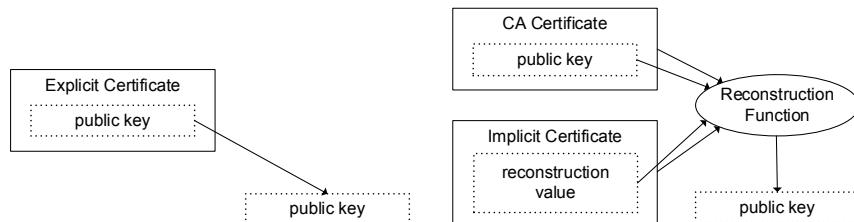
When a certificate is being used to issue certificates it is referred to as a *CA certificate*.

Certificate request permissions are expressed using the same information elements as certificate issuance permissions. When a certificate is being used to request certificates it is referred to as an *enrolment certificate*.

The “cryptographic demonstration that the issuer authorized the linkage between the verification key and the permissions” comes in two forms, referred to as *explicit* and *implicit certificates*.

- If the verification key is explicitly given in the certificate, the certificate is an *explicit certificate*. In this case the cryptographic demonstration that the issuer authorized the linkage is provided by the issuer’s signature on the certificate.
- If the verification key is not explicitly given in the certificate, but is obtained from a *reconstruction value* in the certificate and the issuer’s public key via the reconstruction function specified in 5.3.2, the certificate is an *implicit certificate* and the corresponding verification key is referred to as the *associated public key*. In this case the cryptographic demonstration that the issuer authorized the linkage is provided by the fact that a signature verifies correctly with the verification key that was so derived.

The difference between implicit and explicit certificates is illustrated in Figure 4.



**Figure 4—Implicit and explicit certificates**

An explicit certificate is invalid if it has an implicit certificate as its issuer. An implicit certificate may have an implicit or an explicit certificate as its issuer.

### 5.1.2 Certificate chain

#### 5.1.2.1 Certificate chain construction

A *certificate chain* is a set of certificates ordered from “top” to “bottom”, such that each certificate in the chain is the *issuing certificate* for one below it and the *subordinate certificate* of the certificate above it.

One certificate is the *issuing certificate* for a second one if the certificate holder of the first certificate used the private key of the first certificate to create the final form of the second certificate, either by signing it (in the case of an explicit certificate) or by carrying out cryptographic operations to create a reconstruction value (in the case of an implicit certificate). The counterpart of an issuing certificate is a *subordinate certificate*. If certificate B is the issuing certificate for certificate A, for compactness this certificate uses the terminology “B issues A” even though it would be more correct to use the terminology “B’s holder issues A”.

A *trust anchor* is any certificate that is established to be trustworthy by itself, e.g., by preconfiguration or independent provisioning; in other words, not by reference to any other certificate. A necessary (but not sufficient) condition for a certificate to be valid is that it is possible to construct a certificate chain from the certificate to a trust anchor. The SSME stores information about which certificates are trust anchors.

A *root certificate* is an explicit certificate that is verified with the public key included directly in the certificate, in contrast to other certificates that are verified using the verification key of the issuing

certificate. There is no distinct issuing certificate for a root certificate. All trusted root certificates are by definition trust anchors.

The *length of the chain* is defined as the number of certificates from the certificate under consideration to the root, inclusive of both (so if the root were to sign a SPDUs, the associated certificate chain length would be 1). This is true even if the local trust anchor used by a particular implementation is not the root but some intermediate CA.

To enable construction of certificate chains, each IEEE 1609.2 certificate contains an identifier for its issuer. An example algorithm for the construction of certificate chains is specified in D.3.1 and illustrated in D.3.2. In constructing the certificate chain, the receiver uses a combination of the certificates that were included in the signed SPDUs and locally cached copies of certificates. Local copies of certificates are managed by the SSME.

The issuer identifier is obtained by calculating an eight-octet cryptographic digest of the issuing certificate. There is therefore a  $2^{-64}$  probability that any pair of CA certificates known to the SSME have the same eight-byte digest. If this happens, then when a certificate or SPDUs comes to be validated, two chains can be constructed consistent with the issuer identifiers. The certificate or SPDUs is considered valid if either of the chains is cryptographically valid.

### **5.1.2.2 Maximum supported certificate chain length**

An implementation of WAVE Security Services may have a maximum length of certificate chain that it can support. A conformant implementation shall support a maximum length of at least eight. The Protocol Implementation Conformance Statement (PICS) proforma given in Annex A allows the vendor of an implementation of WAVE Security Services to state the maximum length of certificate chain that the implementation supports.

### **5.1.2.3 Cryptographic validity of a chain**

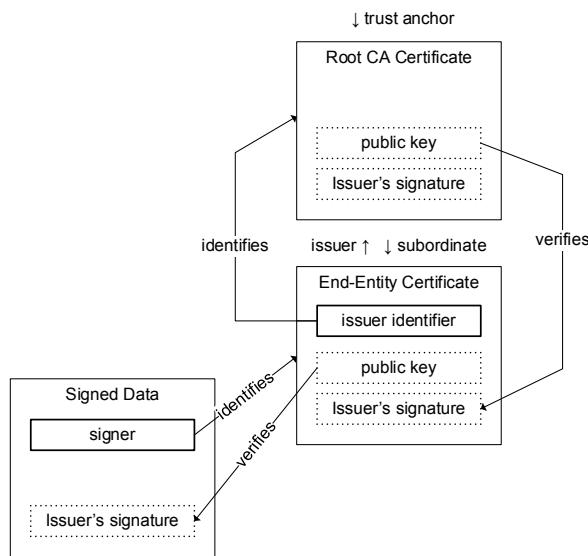
A certificate chain is cryptographically valid if the following conditions hold:

- For each explicit certificate in the chain, the signature on that certificate can be cryptographically verified with the public key in the issuing certificate. See Figure 5 for an illustration.
- If the chain ends with one or more implicit certificate: the signature on a signed SPDUs can be verified with the associated public key from the last certificate in the chain.

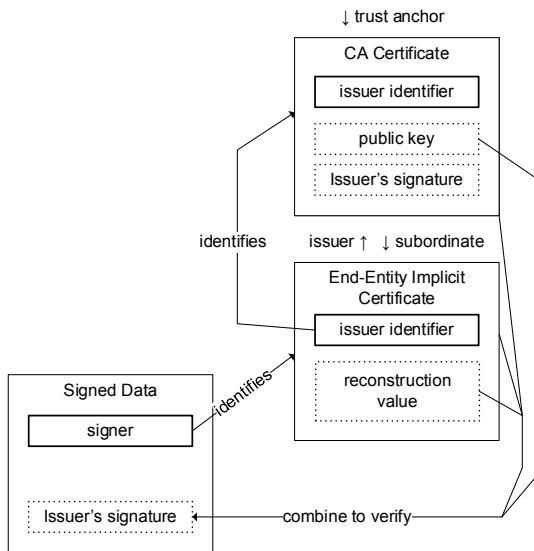
If the chain ends with one or more implicit certificates, and no signed SPDUs can be verified with the associated public key from the last certificate in the chain, then the cryptographic validity of the implicit certificates is undetermined. If an implicit certificate is known to be valid, then an implementation has the option of deriving the public key and caching it, marked as valid, for faster verification of later messages signed with that certificate. If the validity of an implicit certificate is undetermined, this optimization is not available to an implementation because the derived public key cannot be marked as valid.

See Figure 5 for an illustration involving explicit certificates. See Figure 6 for an illustration involving an implicit authorization certificate.

For all signed SPDUs, the hash operation, signature, and verification are carried out as specified in 5.3.1. The encoding of data structures for input to those cryptographic operations is defined in Clause 6.



**Figure 5—Cryptographic verification with explicit certificates**



**Figure 6—Length-2 certificate chain with implicit certificates and a non-root CA as trust anchor**

For further illustrations, see D.3.1.

#### 5.1.2.4 Consistency of permissions within a certificate chain

In a certificate chain associated with a valid signed SPDU, the certificate that signs the PDU includes application permissions and all other certificates include certificate issuance permissions.

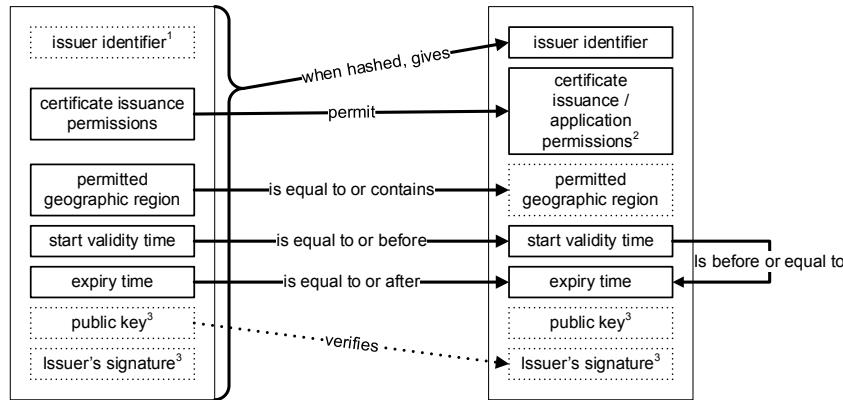
A valid certificate's permissions are consistent with its issuer's permissions. A valid certificate chain contains only valid certificates.

A subordinate certificate is consistent with its issuing certificate if the following conditions hold:

- **Geographic consistency:** No point in the subordinate certificate’s validity region is outside the issuing certificate’s validity region.
- **Validity period consistency:** The subordinate certificate’s validity period is within the issuing certificate’s validity period.
- **Consistency of application/issuance permissions:** The subordinate certificate’s application or certificate issuance permissions are consistent with the issuing certificate’s certificate issuance permissions, i.e., for every (PSID, SSP) entry in the subordinate certificate’s application or certificate issuance permissions (the “subordinate entry”) there is an entry in the issuing certificate’s certificate issuance permissions (the “issuing entry”) such that:
  - The PSID in the subordinate entry is the same as the PSID in the issuing entry.
  - The SSP or SSP Range in the subordinate entry is equal to or a subset of the SSP Range in the issuing entry. See 6.4.34 for a full definition of consistency of SSP with SSP Range, or of one SSP Range with another.
  - If the subordinate entry is for cert issuance or request permissions, the permitted depth of the chain in the subordinate entry is consistent with the permitted depth of the chain in the issuing entry. Specifically, if *minChainDepth* and *chainDepthRange* in the subordinate certificate and issuing certificates have the values  $mcd_s$ ,  $cdr_s$ ,  $mcd_i$ ,  $cdr_i$ , respectively, then  $mcd_i \leq mcd_s + 1$  and  $(mcd_i + cdr_i) \geq (mcd_s + cdr_s + 1)$ . (In the case where the subordinate certificate is an end-entity certificate,  $mcd_s$  and  $cdr_s$  are set equal to zero (0) in these formulas.)
  - The eeType field in the issuing entry permits the subordinate entry, i.e.:
    - If the subordinate entry is in the application permissions field, then:
      - The eeType field in the issuing entry includes the value `app`.
    - If the subordinate entry is in the certificate issuance permissions field, then:
      - All the values of eeType in the subordinate entry also appear in the issuing entry.
- **Cryptographic consistency:**
  - If the issuing certificate is an explicit certificate: the verification key from the issuing certificate can be used to verify the subordinate certificate.
  - If the issuing certificate is an implicit certificate: the cryptographic material in the certificates can be used to verify a signature on a SPDU.

Figure 7 illustrates the process of checking that a subordinate certificate is consistent with its issuing certificate, and additionally captures the process of checking that a certificate is internally consistent as defined in Clause 6. For clarity, only the relevant fields within the data structures are shown. Clause 9 specifies example processing steps that correctly carry out this check.

## Issuing Certificate                      Subordinate Certificate



**NOTES:**

1. Not included if the holder is a root CA
2. Application permissions for the PDU signing certificate, certificate issuance permissions for all other certificates
3. For implicit certificates, the test of cryptographic validity is whether signed data can be cryptographically verified with a public key derived from the issuing certificate and the subordinate certificate.

**Figure 7—Consistency of permissions between an issuing and a subordinate certificate, and within the subordinate certificate**

### 5.1.3 Revocation and expiry

#### 5.1.3.1 General

A certificate is said to be revoked if an appropriately authorized entity states that that certificate is known not to be trustworthy. If a certificate is revoked, the SDS shall consider all SPDUs signed by that certificate and received after the issue date of the revocation statement to be invalid even if their stated generation time is before the issue date of the revocation statement.

If a CA certificate is revoked, the SSME shall indicate that any certificates issued by that CA and first received after the issue date of the revocation statement are revoked, even if their stated generation time is before the issue date of the revocation statement. This applies to any certificate that chains back to the revoked CA certificate.

Information about revoked certificates is stored by the SSME via the SSME-AddRevocationInfo.request and SSME-AddRevocationInfo.confirm primitives. The SSME provides the revocation status of certificates via the SSME-CertificateInfo.request and SSME-CertificateInfo.confirm primitives. Revocation information consists of a series of individual data items and information allowing the SSME to associate the revocation information with specific certificates.

For any certificate  $C$ , there is at most one set of revocation information  $R$  that will indicate that  $C$  is revoked. If  $R$  does not indicate that  $C$  is revoked, then  $C$  is not revoked.

Within the SSME, the process of determining whether or not a certificate is revoked involves two steps:

- a) Determine which set of revocation information applies to the certificate.

- b) Determine whether any individual data item within the relevant revocation information indicates that the certificate is revoked.

The rules used to determine whether a set of revocation information applies to a given certificate are defined in 5.1.3.2.

There are two forms of revocation information. The type of revocation information that applies to a certificate is indicated by the CertificateId field in the certificate:

- *Linkage-based*: If the CertificateId field indicates the choice linkageData, the certificate is revoked by publishing the *linkage seed* value corresponding to the linkageData value. See 5.1.3.4 for a full description.
- *Hash ID-based*: If the CertificateId field indicates the choice name, binaryId, or none, the certificate is revoked by publishing a hash of the certificate. See 5.1.3.5 for a full specification.

All forms of revocation information include the following information fields that indicate which set of certificates it is revoking:

- The *Certificate Revocation Authorizing CA* (CRACA) certificate
- The *CRL Series* value

The use of this information is specified in 5.1.3.2.

The *certificate revocation list* (CRL) is a data structure for distribution of CRL information. A CRL contains one or more revocation information items. The CRL structure and a specification of how to secure CRLs using the mechanisms of this standard are defined in Clause 7.

### **5.1.3.2 Determining which revocation information applies to a given certificate**

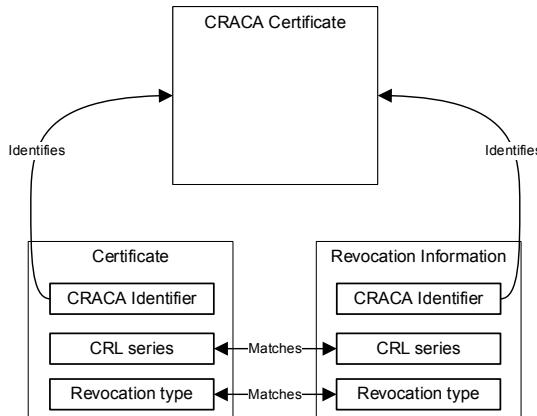
Revocation information applies to a given certificate if it:

- Indicates the same *Certificate Revocation Authorizing CA* (CRACA) certificate, and
- Indicates the same *CRL Series* value, and
- Is of the appropriate type (linkage-based or hash ID-based)

The CRACA is a CA that has authority to authorize the issuance of revocation information for a particular group of other certificates. The CRACA certificate for a certificate  $C$  is only valid if it is on  $C$ 's issuing chain. Likewise, when the revocation information is transported in the form of a signed CRL, the CRACA certificate is only valid if it either signed the CRL itself, or issued the certificate that signed the CRL.

The CRL Series value is an integer that allows a CA to partition its issued certificates into groups that appear on different CRLs.

A certificate indicates the relevant CRACA certificate, CRL series value, and revocation type as specified in 6.4.8. If revocation information is received via a CRL, that CRL indicates the relevant CRACA certificate, CRL series value, and revocation type as specified in 7.3. The primitives SSME-AddRevocationInfo.request, SSME-AddRevocationInfo.confirm, SSME-CertificateInfo.request and SSME-CertificateInfo.confirm allow the CRACA certificate and CRL series to be associated with certificates and revocation information managed by the SSME.

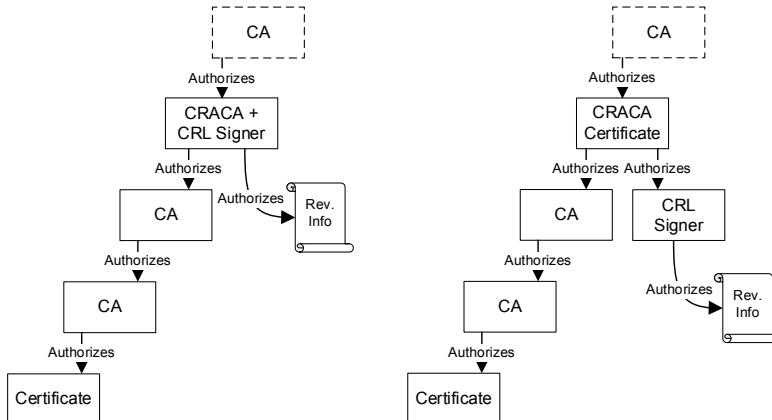


**Figure 8—Relationship between certificate and revocation information**

When revocation information for a given certificate is transported in a signed CRL, the CRL is validly authorized only if one of the following conditions holds:

- The certificate's CRACA signed the CRL, or
- The certificate's CRACA issued a certificate which signed the CRL (referred to as a *CRL signer* certificate)

Examples of the two types of acceptable relationship between the CRL and the CRACA are given in Figure 9.



**Figure 9—Revocation information: issued by a CRACA (on the left) or by a CRL signer directly authorized by the CRACA (on the right)**

### 5.1.3.3 Identification of CRACA certificate

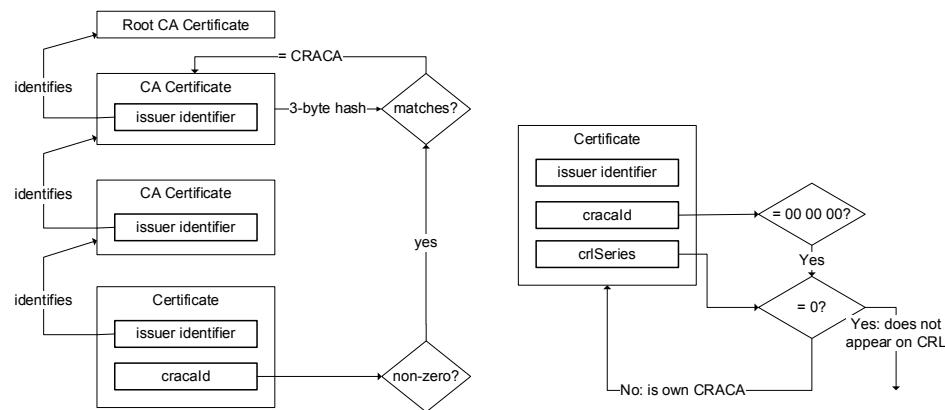
A certificate contains a *cracald* field as specified in 6.4.8. This is an octet string of length 3. The relevant CRACA certificate is the certificate in the chain for which the low-order three bytes of its SHA-256 hash are equal to the *cracald*. The hash of the certificate is obtained as specified in 6.3.25.

A *craclId* of all 0s and a CrlSeries value of 0 indicates that the certificate will not be revoked, i.e., that there is no revocation list that it will appear on. This may be because it has a very short lifetime or for some other reason.

A *craclId* of all 0s and a non-zero CrlSeries value indicates that the certificate will appear on a CRL signed by itself.

If a certificate has a non-zero *craclId*, and the number of certificates in the chain that match the *craclId* is anything other than 1, the certificate is invalid.

Figure 10 illustrates logic flows used in determining the CRACA for a certificate.



**Figure 10—Examples: determining the CRACA certificate for a certificate**

#### 5.1.3.4 Linkage-based revocation information

The purpose of linkage-based revocation information is to allow multiple certificates to be revoked with a single item of revocation information. Linkage-based revocation supports the case where a device or a collection of devices has multiple certificates valid within a time period.

Linkage-based revocation information was originally described in Whyte, et al. [B24].

This standard defines two types of linkage-based revocation information. *Individual linkage information* allows multiple certificates owned by a single device to be revoked. *Group linkage information* allows certificates owned by all devices within a predefined group to be revoked. A certificate may include none, one, or both of individual linkage information and group linkage information. The mechanism by which it is determined at issuance time that different certificates are members of the same group is not specified in this standard.

This subclause specifies how individual and group revocation information is used to determine the revocation status of a certificate containing specific individual or group linkage data.

**Individual linkage data.** A certificate is revoked if it is indicated to be revoked by any of the individual data items within the collection of individual revocation information relevant to the certificate.

A data item within the individual revocation information includes the following information fields. The information fields are provided to the SSME via SSME-AddIndividualLinkageBasedRevocation.request,

SSME-AddIndividualLinkageBasedRevocation.confirm. The specification of these information fields within a CRL is given in 7.3. The use of these fields is explained in this subclause.

- $iRev$ , an integer, an indication of the time period when the revocation information becomes effective
- $iMax$ , an integer, an indication of the time period when the revocation information stops being effective
- $jMax$ , the number of certificates within each time period
- $LinkageSeed1$ , an octet string of length 16
- $LinkageAuthorityIdentifier1$ , an indication of the *linkage authority* that generated linkage seed 1, an octet string of length 2
- $LinkageSeed2$ , an octet string of length 16
- $LinkageAuthorityIdentifier2$ , an indication of the *linkage authority* that generated linkage seed 2, an octet string of length 2

The values  $LinkageSeed1$  and  $LinkageSeed2$  are unique to a particular data item within the revocation information. The other values may be common to multiple data items within the revocation information.

Certificates that include linkage data contain the values indicated in 6.4.10.

- $iCert$ , an indication of the time period that applies to the certificate. The intent of the design is that a given value of  $iCert$  should refer to the same time period for all certificates with the same (CRACA, CRL Series) value.
- $LinkageValue$ , the value used to determine whether or not the certificate is revoked.

The following calculations determine whether a data item indicates that a certificate is revoked. The values  $iRev$ ,  $LinkageSeed1$ , and  $LinkageSeed2$  are modified from their original values during the operation of these algorithms.

- a) If  $iCert > iMax$ , the revocation information is not relevant for this certificate
- b) While  $iRev < iCert$ :
  - 1) Set  $LinkageSeed1$  = the low-order 16 octets of  $[SHA-256(LinkageAuthorityIdentifier1 \parallel LinkageSeed1 \parallel 0^{112})]$ , where  $0^{112}$  is 112 0 bits
  - 2) Set  $LinkageSeed2$  = the low-order 16 octets of  $[SHA-256(LinkageAuthorityIdentifier2 \parallel LinkageSeed2 \parallel 0^{112})]$
  - 3) Set  $iRev = iRev + 1$
- c) For  $j = 0$  to  $jMax - 1$ :
  - 1) Set  $data = LinkageAuthorityIdentifier1 \parallel \text{Uint32}(j) \parallel 0^{80}$ , where  $\text{Uint32}(j)$  indicates  $j$  represented as a 4-octet integer in network byte order
  - 2) Set  $PreLinkageValue1(j) = AES(\text{key} = LinkageSeed1, \text{data} = data) \text{ XOR } (data)$ , where  $LinkageSeed1$  is the value  $LinkageSeed1$  takes after completing the iterative process defined in step b)
  - 3) Set  $PreLinkageValue2(j) = AES(\text{key} = LinkageSeed2, \text{data} = [0^{10} \parallel LinkageAuthorityIdentifier2 \parallel \text{Uint32}(j)]) \text{ XOR } [0^{10} \parallel LinkageAuthorityIdentifier2 \parallel \text{Uint32}(j)]$
  - 4) Set  $LinkageValue(j) =$  the low-order 9 bytes of  $PreLinkageValue1$  XOR  $PreLinkageValue2$
  - 5) If  $LinkageValue(j) = LinkageValue$ , the certificate is considered revoked

**Group linkage data.** A certificate is revoked if any data item within the collection of group revocation information relevant to the certificate indicates that the certificate is revoked.

A data item within the group revocation information includes the following information fields. The information fields are provided to the SSME via SSME-AddGroupLinkageBasedRevocation.request and SSME-AddGroupLinkageBasedRevocation.confirm. The specification of these information fields within a CRL is given in 7.3. The use of these fields is explained in this subclause.

- $iRev$ , an integer, indication of the time period when the revocation information was generated
- $iMax$ , an integer, indication of the time period when the revocation information stops being effective
- $GroupLinkageSeed1$ , an octet string of length 16
- $LinkageAuthorityIdentifier1$ , an indication of the *linkage authority* that generated linkage seed 1, an octet string of length 3
- $GroupLinkageSeed2$ , an octet string of length 16
- $LinkageAuthorityIdentifier2$ , an indication of the *linkage authority* that generated linkage seed 2, an octet string of length 3

The values  $GroupLinkageSeed1$  and  $GroupLinkageSeed2$  are unique to a particular data item within the revocation information. The other values may be common to multiple data items within the revocation information.

Certificates that include group linkage data contain the values indicated in 6.4.12:

- $iCert$ , an indication of the time period that applies to the certificate. The intent of the design is that a given value of  $iCert$  should refer to the same time period for all certificates with the same (CRACA, CRL Series) value.
- $j$ , the index of the certificate within the current time period.
- $GroupLinkageValue$ , the value used to determine whether or not the certificate is revoked.

The following calculations determine whether a data item indicates that a certificate is revoked:

- a) If  $iCert > iMax$ , the revocation information is not relevant for this certificate.
- b) While  $iRev < iCert$ :
  - 1) Set  $GroupLinkageSeed1$  = the low-order 16 octets of  $[SHA-256(LinkageAuthorityIdentifier1 \parallel GroupLinkageSeed1 \parallel 0^{112})]$ , where  $0^{112}$  is 112 0 bits
  - 2) Set  $GroupLinkageSeed2$  = the low-order 16 octets of  $[SHA-256(LinkageAuthorityIdentifier2 \parallel GroupLinkageSeed2 \parallel 0^{112})]$
  - 3) Set  $iRev = iRev + 1$
- c) Set  $data = LinkageAuthorityIdentifier1 \parallel \text{Uint32}(j) \parallel 0^{80}$ , where  $\text{Uint32}(j)$  indicates  $j$  represented as a 4-octet integer in network byte order.
- d) Set  $PreLinkageValue1 = AES(\text{key} = GroupLinkageSeed1, \text{data} = data) \text{ XOR } data$ , where  $GroupLinkageSeed1$  is the value  $GroupLinkageSeed1$  takes after completing the iterative process defined in step b)
- e) Set  $PreLinkageValue2 = AES(\text{key} = GroupLinkageSeed2, \text{data} = [LinkageAuthorityIdentifier2 \parallel \text{Uint32}(j) \parallel 0^{80}]) \text{ XOR } [LinkageAuthorityIdentifier2 \parallel \text{Uint32}(j) \parallel 0^{80}]$
- f) Set  $GroupLinkageValue =$  the low-order 9 bytes of  $PreLinkageValue1 \text{ XOR } PreLinkageValue2$
- g) If  $GroupLinkageValue = GroupLinkageValue$ , the certificate is considered revoked

### 5.1.3.5 Hash ID-based revocation information

For certificates that do not include linkage data, there is no group revocation information, only individual revocation information. Individual or hash ID-based revocation information is provided to the SSME via SSME-AddHashIdBasedRevocation.request and SSME-AddHashIdBasedRevocation.confirm. An individual revocation information item consists of:

- *certId*, an octet string of length 10

The following calculations determine whether a data item indicates that a certificate is revoked:

- a) Set *tmpCertId* equal to the HashedId10 of the certificate.
- b) If *tmpCertId* = *certId*, the certificate is considered revoked.

### 5.1.3.6 Dubious certificates

For each known (CRACA, CRL Series) pair, the SSME maintains an *expected update* time, i.e., the time when the revocation information issuer has indicated that revocation information is going to be updated. This value is set to “undefined” if the SSME has never received revocation information for that (CRACA, CRL Series) pair. The expected update time for revocation information contained in a CRL is given in the *nextCrl* field.

A certificate is considered by the SSME to be a *dubious certificate* if either no revocation information is available for that certificate, or the expected update time for that revocation information is in the past.

If queried about the revocation status of a dubious certificate via SSME-CertificateInfo.request, the SSME indicates that the certificate is dubious via SSME-CertificateInfo.confirm.

Any certificate in the chain associated with a signed SPDU might potentially be dubious. The primitive Sec-SecureDataPreprocessing.confirm indicates the earliest *nextCrlTime* associated with any certificate in the chain associated with a signed SPDU. If that time is in the past, the certificate is considered dubious.

The standard provides the following mechanisms to handle the case where the SDS determines that a SPDU signed with a dubious certificate would be valid if the certificate was known not to be revoked, i.e., it passes all checks except that its revocation status is undetermined:

- **Use Overdue CRL Tolerance within SDS:** The SDS may be passed a parameter Overdue CRL Tolerance via Sec-SignedDataVerification.request. In this case, if the earliest *nextCrl* time for any certificate in the chain is in the past by more than Overdue CRL Tolerance, the SDS indicates that the signed SPDU is invalid. If the parameter is not passed, the SDS indicate as valid a signed SPDU that meets all other validity conditions, regardless of the *nextCrl* time values.
- **SDEE-specific processing:** The SDEE may alternatively obtain the earliest *nextCrl* time for any certificate in the chain via Sec-SecureDataPreprocessing.request, Sec-SecureDataPreprocessing.confirm. How a SDEE handles dubious certificates is SDEE-specific.

## 5.2 Signed SPDU validity

### 5.2.1 General

A signed SPDU for a SDEE is valid for use by a receiving SDEE if all of the following hold:

- The SPDU meets a set of conditions that depend only on information sent by the sender, referred to as *consistency conditions*. These are discussed in 5.2.3.
- The SPDU meets other criteria, referred to as *relevance conditions*, which take into account the local time, location, and other state of the receiving SDEE. These are discussed in 5.2.4.
- The SPDU contains no unsupported *critical information fields*. Critical information fields are information fields that are necessary to determine whether a SPDU is valid. This is discussed in 5.2.5.

Consistency conditions make use of the claimed generation time and location of the signed SPDU. Relevance conditions make use of the claimed generation time and location of the signed SPDU, and the current time and location of the receiving SDEE. Time and location measurement requirements for the SDS are discussed in 5.2.2.

### 5.2.2 Local estimates of time and location

Some of the consistency and relevance conditions within WAVE Security Services make use of estimates of time and location. Estimates of time and location (first-order statistical information) are assumed to be provided by a location server such as a Global Navigation Satellite System (GNSS) subsystem.

When generating a signed SPDU, the estimated generation time may be included. The estimated generation time is an estimate of the time at which the information elements for the signed SPDU were assembled for encoding.

When carrying out relevance tests on signed SPDUs, secure data exchange services use estimated generation time as described in 5.2.4.2.2, 5.2.4.2.3, and 5.2.4.2.5.

When generating a signed SPDU, the estimated generation location may be included. The estimated generation location is an estimate of the location of the transmitter at the estimated generation time defined above. When carrying out relevance tests on signed SPDUs, secure data exchange services use the estimated generation location estimates as described in 5.2.4.2.5.

This standard only specifies the use of first-order statistics in performing consistency and relevancy checks. More sophisticated relevance checks, including ones using second-order statistics to account for estimation error (co-)variances, are SDEE-specific. Performance requirements on time and location estimation are out of scope of this standard.

### **5.2.3 Consistency conditions**

#### **5.2.3.1 General**

There are two types of consistency conditions; *global consistency conditions* which do not depend on the specific SDEE that consumes the SPDU, and *SDEE-specific consistency conditions* which do depend on the receiving SDEE but not on its local conditions.

#### **5.2.3.2 Global consistency conditions**

##### **5.2.3.2.1 General**

Global consistency conditions are:

- The PDU is correctly formed using the data structures of Clause 6.
- There is a certificate chain that leads from the signing certificate to a known trust anchor, constructed as specified in 5.1.2.1, such that:
  - All of the certificates in the chain are correctly formed using the data structures of Clause 6.
  - All certificates in the chain pass cryptographic verification with the appropriate public keys as specified in 5.1.2.3.
  - The certificate chain is internally consistent as specified in 5.1.2.4.
  - None of the certificates in the chain have been revoked as specified in 5.1.3.
- The PDU is consistent with the signing certificate:
  - The permissions indicated by the security envelope are consistent with the permissions in the signing certificate, and the security envelope is consistent with itself, as specified in 5.2.3.2.2.
  - The PDU cryptographically verifies with the appropriate public keys. The cryptographic operations used for signing and verification are specified in 5.3.1. The encoding of data structures for input to those cryptographic operations is defined in Clause 6.
- The PDU is internally consistent, as specified in 5.2.3.2.3.

If a signed SPDU does not meet all of these conditions it is invalid.

##### **5.2.3.2.2 Consistency between signed SPDU and signing certificate**

A signed SPDU contains the following information elements that are used when determining validity:

- Required:
  - Identifier of signing certificate
  - Associated PSID
  - One of: Encapsulated payload or hash of external payload
- Optional:
  - Generation location (see 5.2.2)

- Generation time (see 5.2.2)
- Expiry time

The contents of a signed SPDU are fully specified in Clause 6.

The certificate used to sign a PDU is identified using the SignerIdentifier structure within the SignedData as specified in Clause 6. This is one of: an identifier of the signing certificate, the signing certificate itself, or a certificate chain including the end-entity certificate and a series of issuing certificates up to, but not including, the root. If the SPDU contains an identifier of the signing certificate, the receiving SDS can only determine validity of the SPDU if the certificate is locally stored and managed by the SSME.

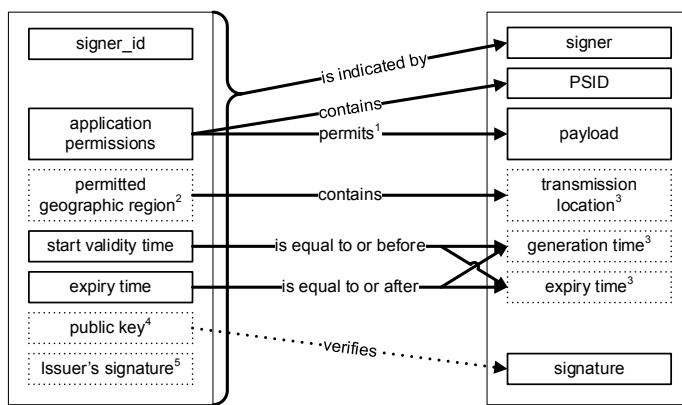
A signed SPDU is consistent with the signing certificate if all the following hold:

- The signing certificate is an authorization certificate, i.e., it contains application permissions.
- The stated generation location is consistent within the geographic region indicated in the certificate, i.e., one of the following conditions holds:
  - Either the certificate is valid worldwide.
  - Or the certificate has a geographic restriction, the SDEE specification states that the signed SPDU contains a generation location, and the generation location is within the geographic restriction.
  - Or the certificate has a geographic restriction but the SDEE specification states that the SPDU generation location is not used for consistency checks. (This can be stated using the 1609.2 security profile specified in Annex C.)
- The stated generation time is within the validity period of the certificate.
- The expiry time, if present, is within the validity period of the certificate.
- The PSID that appears in the security envelope of the signed SPDU appears in the *appPermissions* field of the certificate.
- The public key in or associated with the certificate can be used to cryptographically verify the signature on the PDU.

A signed SPDU that is inconsistent with its signing certificate is invalid. A signed SPDU that is consistent with its signing certificate is valid so long as the other validity conditions specified in this standard are satisfied.

Figure 11 illustrates the necessary conditions for signed data to be consistent with the associated signing certificate. The boxes within the data and certificate identify the information elements that are conveyed by those data structures. For clarity, only the relevant fields within the data structures are shown. See Clause 9 for specification of processing that correctly performs these checks.

## Signing Certificate                      Signed PDU



NOTES:

1. Determined using the PSID and SSP. The process to determine whether the operational permissions permit the message payload is specified by the organization reserving the PSID and is out of scope for this standard.
2. Included per policy set by the appropriate authority for the region where the certificate is being used.
3. Optional. Inclusion of this data is as determined by the organization reserving the PSID. This data may be contained in the payload or within the security header fields.
4. For implicit certificates, the public key is derived rather than explicitly stated within the certificate.
5. Not included in an implicit certificate.

**Figure 11—Consistency of permissions between signed SPDU and signing certificate**

### 5.2.3.2.3 Internal consistency in signed SPDU

If a signed SPDU contains a generation time and an expiry time, the PDU is inconsistent with itself and hence invalid if the generation time is after the expiry time.

### 5.2.3.3 SDEE-specific consistency conditions

#### 5.2.3.3.1 General

SDEE-specific consistency conditions are:

- The Provider Service ID (PSID) in the SPDU is consistent with any other PSID that the SDEE associates with the received PDU as specified in 5.2.3.3.2. This condition is checked by Sec-SignedDataVerification.request if the SDEE so requests and provides the appropriate PSID in that request.
- The payload of the PDU is consistent with the permissions (PSID, SSP, assurance level) in the signing certificate as specified in 5.2.3.3.3. This condition cannot be verified by the SDS and is intended to be verified by the receiving SDEE.
- Any external data included in the calculation of the signature has the correct hash value as specified in 5.2.3.3.4. This condition cannot be verified by the SDS and is intended to be verified by the receiving SDEE.
- (Optionally) The number of certificates in the chain, from end-entity certificate to root certificate inclusive, is less than some SDEE-specific limit (see Annex C). This condition can be verified by the SDS.

If a signed SPDU does not meet all of these conditions it is invalid.

### **5.2.3.3.2 Consistency between PSID in signed SPDU and PSID derived from context**

Depending on the context in which a signed SPDU is received, a SDEE might have access to metadata which it can use to associate that signed SPDU with a PSID. As examples: (1) if the signed data was received within the WAVE Short Message Protocol (WSMP) specified in IEEE Std 1609.3 with TPID = 0 or 1, the WSMP header includes a PSID; (2) the SDEE might be associated with a one and only one PSID and therefore can associate all received PDUs with that PSID. This PSID is referred to as a *PSID derived from context*.

As illustrated in Figure 11, the IEEE 1609.2 signed SPDU structure explicitly states the PSID with which the creator intends the signed SPDU to be associated. This is referred to as the *transmitted PSID*.

A received signed SPDU is invalid unless the transmitted PSID is the same as the PSID derived from context. The Sec-SignedDataVerification.request primitive specified in Clause 9 supports carrying out this check within the SDS, even though it is SDEE-specific: the invoking SDEE provides the PSID derived from context and the signed SPDU, and the SDS checks that the PSID derived from context is identical to the transmitted PSID from the signed SPDU. In another possible implementation, this check may be realized directly by the SDEE.

### **5.2.3.3.3 Consistency between SPDU payload and permissions: Service Specific Permissions**

A valid signed SPDU satisfies the following conditions that address consistency of the PDU payload with the sender's permissions.

- The PDU payload is consistent with the PSID in the security envelope.
- The PDU payload is consistent with the relevant Service Specific Permissions (SSP) in the authorization certificate, if any.
- The PDU payload is consistent with the assurance level in the authorization certificate, if any.

Consistency between the payload and the certificate is SDEE-specific and outside the scope of this standard. Consistency should be defined as part of the specification of the SDEE (i.e., the SDEE specification should provide a map from the SSP values to the payload fields and their values that are permitted by each SSP value) and checks for consistency should be implemented by the receiving SDEE.

The PSID, SSP, and assurance level are contained within the certificate as specified in 6.4.8. The certificate format supports inclusion of multiple (PSID, SSP) pairs, but no PSID appears more than once in a valid certificate, so the correct PSID can be unambiguously associated with a signed SPDU.

The Sec-SecureDataPreprocessing.request returns the PSID, SSP, and assurance level associated with a signed SPDU.

For discussion of the use of the SSP and assurance level and the responsibilities of a PSID owner, see Annex C.

### **5.2.3.3.4 External data**

WAVE Security Services support generating a signed SPDU in which the signature calculation includes data that is only indirectly included in the payload of the signed SPDU. In this case, the hash of the external data is included. The signed SPDU is valid only if the external data indicated hashes to the value included in the signed SPDU. How this data is defined, obtained, and shared between the sending and receiving SDEEs is outside the scope of this standard.

## 5.2.4 Relevance conditions

### 5.2.4.1 General

The relevance conditions that apply to a received PDU are SDEE-specific. Relevance conditions are:

- *Security service-verified relevance conditions*: Relevance conditions that can be tested within the SDS, such as whether the PDU is a replay of a previously received PDU, whether its generation time is sufficiently recent, or whether its generation location is sufficiently local. See 5.2.4.2 for further discussion.
- *SDEE-verified relevance conditions*: Other relevance conditions based on SDEE-specific criteria that cannot be tested within the SDS. See 5.2.4.3 for further discussion.

### 5.2.4.2 SDS-verified relevance conditions

#### 5.2.4.2.1 General

The following relevance conditions depend on the local state of the receiving SDEE and can be checked by the SDS. Whether or not any or all of these relevance conditions apply—and if they apply what parameters are used with them—is SDEE-specific and is intended to be part of the SDEE specification. The 1609.2 security profile is provided to enable SDEE specifiers to specify the relevance conditions that apply (see Annex C). The relevance conditions are specified in more detail in subsequent subclauses. A signed SPDU is valid only if it is valid with respect to each specified relevance condition as defined in the appropriate subclause below.

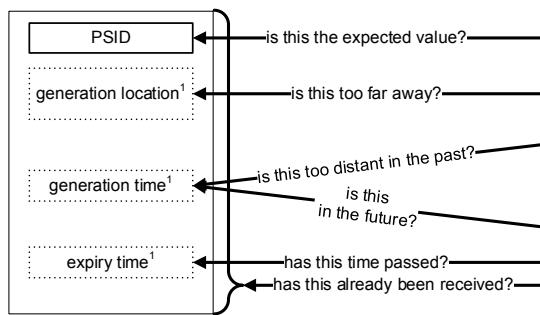
- **Future generation**: The signature generation time is not in the future.
- **Freshness**: The signature generation time is not too long ago (for some PSID-specific definition of “not too long ago”).
- **Expiry**: The signed data has not expired.
- **Location**: The generation location is within a location tolerance  $t_l$  of the receiver’s location.
- **Replay**: The PDU is not a duplicate of a PDU acted upon by that SDEE in the recent past (for some PSID-specific definition of “the recent past”).
- **Certificate expiry**: None of the certificates in the chain leading to the signed data have expired.

The data structures defined in Clause 6 allow the information necessary to be transported either in the payload of the signed data or in the security envelope.

The SDS supports testing against all of these conditions. The Sec-SignedDataVerification.request primitive (see 9.3.12.1) allows the invoking SDEE to specify which of the conditions apply and the tolerance values associated with freshness and location if those conditions are to be tested. Since replay detection pertains to the same SPDU being processed twice by a single SDEE, the Sec-SignedDataVerification.request primitive makes use of a SDEE identifier, which is assigned and managed by the SDS as defined in 9.2.1.

Figure 12 illustrates the fields to which relevance tests apply (the check for expired certificates is not illustrated in Figure 12). The relevance and replay tests to be carried out are specified as inputs to Sec-SignedDataVerification.request.

## Signed Data



NOTES:

1. This data may be contained in the payload or within the security header fields.

**Figure 12—Replay and relevance tests**

### **5.2.4.2.2 Generation time too far in the past**

The following algorithm is defined to determine whether the generation time of a signed SPDU is too far in the past, for some SDEE-specific (and possibly local condition-specific) definition of “too far”.

The difference between the local estimate of time at which the SPDU was received and the estimated generation time contained in that SPDU is calculated. If that difference exceeds  $V$ , the validity period associated with PDUs of the same type as that received, the PDU is invalid. Otherwise, the PDU is valid with respect to this relevance condition.

### **5.2.4.2.3 Generation time in the future**

The following algorithm is defined to determine whether the generation time of a signed SPDU is too far in the future, for some SDEE-specific (and possibly local condition-specific) definition of “too far”.

The difference between the local estimate of time at which the SPDU was received and the estimated generation time contained in the SPDU is calculated. If that difference is less than zero, the PDU is invalid. Otherwise, the PDU is valid with respect to this relevance condition.

### **5.2.4.2.4 Expiry time**

The following algorithm is defined to determine whether a signed SPDU should be considered expired.

The difference between the local estimate of time at which the SPDU was received and the expiry time contained in that SPDU is calculated. If that difference is greater than zero, the PDU is invalid. Otherwise, the PDU is valid with respect to this relevance condition.

### **5.2.4.2.5 Generation location too distant**

The following algorithm is defined to determine whether the estimated generation location of the message is too far away.

The distance between the estimated generation location of the signed SPDU and the receiver's estimated location is calculated. If this distance is greater than D, the rejection threshold distance, the PDU is invalid. Otherwise, the PDU is valid with respect to this relevance condition.

#### **5.2.4.2.6 Replay**

The SDS can be requested to detect whether a signed SPDU is an exact duplicate of one previously processed by the SDS for that SDEE. The replay detection service is provided by SSME-Sec-ReplayDetection.request, SSME-Sec-ReplayDetection.confirm. The replay detection service indicates that a signed SPDU is a replay if the entire encoded signed SPDU, including signature and other fields such as generation time inserted by the SDS, is identical to a recently received SPDU. The definition of "recently received" is SDEE-specific, but it is a logically consistent choice for this value to be the same as the value used to determine whether a SPDU has a generation time too far in the past (see 5.2.4.2.2), and the interfaces defined in this standard enforce that the two values are the same. Other replay detection techniques, such as ones based on the payload only or on the same data encoded in different ways, are out of scope of this standard.

Whether or not the SDS carries out replay detection, and the length of the interval for replay detection, is part of the SDEE specification. The 1609.2 security profile (see Annex C) is provided to specify replay detection behavior by the security services. Annex C also provides discussion of how to establish whether replay detection is important for a particular SDEE.

#### **5.2.4.2.7 Certificate expiry**

The following algorithm is defined to determine whether a signed SPDU should be considered expired.

The pairwise difference between the local estimate of time at which the SPDU was received and the expiry time in each certificate in the chain that signed that SPDU is calculated. If any of those differences is greater than zero, the PDU is invalid. Otherwise, the PDU is valid with respect to this relevance condition.

Certificates that have expired are risky to trust, since they are not guaranteed to appear on a CRL. It is strongly recommended that a signed SPDU received after the expiry time of any certificate in its chain be rejected.

The 1609.2 security profile (see Annex C) is provided to specify whether certificate expiry is checked by the security services. Annex C also provides discussion of how to establish whether certificate expiry detection is important for a particular SDEE.

#### **5.2.4.3 SDEE-verified relevance conditions (informative)**

In addition to the relevance conditions that can be checked by the SDS, a SDEE specification could include other relevance conditions (for example, PDUs whose sender is going to be out of range shortly, based on their stated location and velocity, might be rejected; or the recent generation time condition could be checked using just the local time estimate rather than the probability distribution function of the local time; or the distance condition might use a more sophisticated distribution for the location). The SDS do not check these relevance conditions and they do not need to be specified in the 1609.2 security profile for the PSID (see Annex C).

### 5.2.5 Supported critical information fields

Critical information fields are any fields necessary to establish the validity of a signed SPDU. An implementation of WAVE Security Services that cannot parse critical information fields in a signed SPDU or a certificate shall consider that signed SPDU or certificate to be invalid.

An implementation of WAVE Security Services might not be able to parse critical information fields for a number of reasons, including:

- The fields are too long.
- An array contains too many entries.
- A recursive structure contains too many recursions.
- A structure that uses identifiers includes an identifier that the implementation does not recognize.

For each data type defined in Clause 6 that may be of arbitrary length (in octets or number of entries) or depth, the definition in Clause 6 specifies the circumstances under which it is a critical information field, and a minimum size to be supported by any conformant implementation of WAVE Security Services. The Protocol Implementation Conformance Statement (PICS) provided in Annex A allows an implementation to state any size it supports beyond the minimum required for conformance.

## 5.3 Cryptographic operations

### 5.3.1 Signature algorithms

This standard specifies use of the Elliptic Curve Digital Signature Algorithm (ECDSA) specified in Federal Information Processing Standard (FIPS) 186-4, optionally with the inclusion of additional information in the signature as specified in SEC 1 Version 2.<sup>9</sup>

Two elliptic curves are specified for use with ECDSA: NIST P-256 as specified in FIPS 186-4, and brainpoolP256r1 as specified in RFC 5639. Data structures and encoding rules for data objects associated with ECDSA are specified in Clause 6 of this standard and include an indication of which curve is applicable.

When data is hashed for signing or verification, the hash shall be created using the following rules:

- a) The hash algorithm shall be one of the algorithms defined in 5.3.3.
- b) The data shall have an assigned *verification type*, which is either *certificate*, indicating that the message is to be verified with a certificate, or *self-signed*, indicating that it is to be identified with a key embedded in the message itself (in this second case the message is also called “self-signed”).
- c) The hash value input to the signing or verification process shall be:

Hash ( Hash (*Data input*) || Hash (*Signer identifier input*) )

where

- 1) *Data input* = the data to be signed
- 2) *Signer identifier input* depends on the verification type of the message

---

<sup>9</sup> The additional information may allow faster verification operations as described in SEC1 Version 2 and Antipa, et al. [B2].

- i) If the verification type is *certificate*, *signer identifier input* shall be the certificate with which the message is to be verified.
- ii) If the verification type is *self-signed*, *signer identifier input* shall be the empty string, i.e., a string of length 0.

The encoding of data for input to the hash process for signing PDUs is specified in 6.3.4.

The encoding of the certificate for input to the hash process is specified in 6.4.3 and 6.4.8.

The choice of verification type, and how it is indicated to a receiver, is specified in 6.3.24.

The encoding of the signature is specified in 6.3.28.

A primitive indicating the information that is passed to the signing operation is given in 9.3.9.1.

NOTE—This processing is different from that specified in Std 1609.2-2013: the hash input to the signature and verification algorithm is calculated differently. Signatures calculated according to the process in this standard do not verify with implementations of IEEE Std 1609.2-2013, and vice versa.<sup>10</sup>

### 5.3.2 Implicit certificates

Implicit certificates were proposed in Brown, Gallant, and Vanstone [B3] and Pintsov and Vanstone [B18], and modifications to protect against attacks were proposed in Brown, Campagna, and Vanstone [B4]. In this standard, implicit certificates are processed as specified in Standards for Efficient Cryptography (SEC) 4 except for the exceptions noted in this subclause.

- a) In this standard, an implicit certificate is encoded as an ImplicitCertificate, as defined in 6.4.5, encoded with the Canonical Octet Encoding Rules (COER). All references to “the certificate CertU” in SEC 4 should be taken as referring to the encoded ImplicitCertificate except when the implicit certificate is hashed to an integer modulus n; this case is addressed in item b) below.
- b) When an implicit certificate is hashed to an integer modulo n, the input is not simply the implicit certificate CertU but the information specified below. This affects the following steps in SEC 4:
  - 1) Section 3.4, Action 7
  - 2) Section 3.5, Action 4
  - 3) Section 3.6, Action 2
  - 4) Section 3.7, Action 4
  - 5) Section 3.8, Action 4

The encoded data input to the hash function is Hash (ToBeSignedCertificate from the subordinate certificate as specified in 6.4.8) || Hash (Entirety of issuer certificate as specified in 6.4.3).

SHA-256 shall be used as the Hash algorithm H within the integer hash Hn specified in SEC 4, section 2.3.

The private key is judged as valid or invalid relative to an implicit certificate using the techniques of SEC 4 section 3.6.

---

<sup>10</sup> Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

### **5.3.3 Hash algorithms: SHA-256**

The only hash algorithm approved for use in this standard is SHA-256 as specified in the Federal Information Processing Standard (FIPS) 180-4. In this standard, the phrase “the SHA-256 hash of [an octet string]” is used to mean “the hash of [that octet string] obtained using SHA-256 as specified in FIPS 180-4”.

### **5.3.4 Encrypted data**

The SDS generates encrypted data in one of two ways, the ephemeral data encryption key approach or the static data encryption key approach.

#### **5.3.4.1 Ephemeral data encryption key approach**

In the ephemeral data encryption key approach:

- The plaintext  $P$  has the form of a valid encoded Ieee1609Dot2Data structure.
- $P$  is encrypted with a freshly and randomly generated symmetric data encryption key  $k$  for an approved symmetric algorithm as specified in 5.3.8, to obtain a ciphertext  $C$ . If the approved symmetric algorithm uses a nonce, that nonce is generated freshly and at random.
- The ciphertext  $C$  is encoded as a SymmetricCiphertext.
- For each recipient key:
  - The data encryption key  $k$  is encrypted with the recipient key to obtain an encrypted data encryption key  $ek$ .
  - The encrypted data encryption key  $ek$  is encoded in a RecipientInfo of the type determined by the type of the recipient’s encryption key as specified in 6.3.31.
- The RecipientInfo structures and SymmetricCiphertext are encoded in an EncryptedData structure which in turn is encoded in an Ieee1609Dot2Data structure.

A single input PDU is encrypted for one or more public or symmetric keys, resulting in a single encrypted PDU that may be decrypted by the holder or holders of the decryption key corresponding to any of the encryption keys. Each of the encryption keys used is referred to as a *recipient key* and the owner of the corresponding decryption key is referred to as a *recipient*. A public key used for encryption could have been obtained by the encrypter from the recipient’s certificate, from the `encryptionKey` field in a SignedData, or by some other means. If a certificate does not contain an encryption key, it cannot be used to encrypt a PDU for its holder. This standard does not specify how an encrypter obtains a symmetric key to use for encryption.

The cryptographic processing for encryption with the public key varies depending on the source of the public key; see 5.3.5 for details.

If the recipient key is a symmetric key, the data encryption key is encrypted with the symmetric key as specified in 5.3.8 and the relevant RecipientInfo is of type SymmRecipientInfo.

#### **5.3.4.2 Static data encryption key approach**

In the static data encryption key approach:

- The plaintext  $P$  has the form of a valid encoded Ieee1609Dot2Data structure.

- P is encrypted with a previously agreed symmetric data encryption key  $k$  for an approved symmetric algorithm as specified in 5.3.8, to obtain a ciphertext  $C$ . If the approved symmetric algorithm uses a nonce, that nonce is generated freshly and at random.
- The ciphertext  $C$  is encoded as a SymmetricCiphertext.
- An indicator of the key  $k$  is included in a RecipientInfo of type PreSharedKeyRecipientInfo.
- The RecipientInfo and SymmetricCiphertext are encoded in an EncryptedData structure which in turn is encoded in an Ieee1609Dot2Data structure.

### 5.3.5 Public key encryption algorithms: ECIES

The only asymmetric encryption algorithm specified in this standard is the Elliptic Curve Integrated Encryption Scheme (ECIES) as specified in IEEE Std 1363a.

Two elliptic curves are specified for use with ECIES: NIST P-256 as specified in FIPS 186-4, and brainpoolP256r1 as specified in RFC 5639. Data structures and encoding rules for data objects associated with ECIES are specified in Clause 6 of this standard and include an indication of which curve is applicable.

In this standard, ECIES is used to encrypt symmetric keys. Data is encrypted with AES-CCM (see 5.3.8). References to the “data encryption method” below are for consistency with IEEE Std 1363a.

When encrypting with ECIES, the following constraints on the specification in IEEE Std 1363a shall be applied:

- The public key  $V$  shall be freshly generated for each encryption operation.
- The secret value derivation primitive shall be Elliptic Curve Secret Value Derivation Primitive–Diffie-Hellman version with cofactor multiplication (ECSVDP-DHC).
- The data encryption method shall be a stream cipher based on Key Derivation Function 2 (KDF2) which shall be parameterized by the choice:
  - $Hash = SHA-256$
  - $P1$ : recipient information, see below
- The data authentication code shall be MAC1 which shall be parameterized by the choices:
  - Input key length = 256 bits
  - $Hash = SHA-256$
  - $tBits = 128$
  - $P2 =$  the empty string
- Encryption shall use non-DHAES mode. This means that the elliptic curve points shall be converted to octet strings using LSB compressed representation.

The parameter  $P1$  is a hash of the information that was bound to the ECIES key used for the encryption:

- If the encryption key was obtained from a certificate  $c$ ,  $P1$  is SHA-256 ( $c$ ). The certificate shall be put in canonicalized form when hashing: see 6.4.3.
- If the encryption key was obtained from a SignedData within an Ieee1609Dot2Data  $d$  (i.e., the encryption key is `d.signedData.tbsData.headerInfo.encryptionKey.public`),  $P1$  is SHA-256 ( $d$ ).

- If the encryption key was obtained from a different source, P1 is SHA-256 (“”, the empty string).

How a SDEE obtains encryption keys, and which form the parameter  $P1$  takes, is SDEE-specific.

The output of this encryption is a triple  $(V, C, tag)$ , where:

- $V$  is an octet string representing the sender’s ephemeral public key.
- $C$  is the encrypted symmetric key.
- $tag$  is the authentication tag.

NOTE—The case where P1 is the hash of the empty string is defined only for use in responses to anonymous certificate requests and is not recommended for use in any other case. It should only be used if the SignedData is not available as it potentially allows misbinding attacks.

### 5.3.6 Key pair generation

Key pairs for ECDSA or ECIES shall be generated according to the specification in Annex B.4 of FIPS 186-4. Implementations should use a high-quality random number generator to generate the key pair such as those used by NIST [B15].

### 5.3.7 Key pair validity

For ECDSA and ECIES, a key pair is judged to be valid or invalid relative to the criteria in subclause 7.1.3 and Annex A.16.10 of IEEE Std 1363<sup>TM</sup>-2000.

### 5.3.8 Symmetric algorithms: AES-CCM

The only symmetric algorithm specified for use in this standard is the Advanced Encryption Standard (AES) in Counter Mode with Cipher Block Chaining Message Authentication Code (CCM) mode as specified in National Institute for Standards and Technology (NIST) Special Publication (SP) 800-38C.

The ciphertext shall be calculated according to the specification of AES-CCM in NIST SP 800-38C.

The formatting mechanism used shall be the one specified in Appendix A.2 of NIST SP 800-38C, with the following specific choices:

**Control information and nonce (A.2.1):** There is no associated data, so  $Adata = 0$ . The message authentication code length  $Tlen$  shall be 128 bits (16 octets). The octet length of the nonce  $N$  shall be 12, leaving three octets to encode the length of the data. The nonce is generated freshly at random for every invocation of AES-CCM to encrypt.

**Formatting of the associated data (A.2.2):** There shall be no associated data.

The counter block generation mechanism used shall be the one specified in Appendix A.3 of NIST SP 800-38C.

The input to AES-CCM encryption with no associated data is the nonce  $N$  and the payload  $P$  of length  $Plen$  bits. The output is the ciphertext  $C$  of length  $Clen = Plen + Tlen$  bits.

On decryption using the mechanisms of NIST SP 800-3C, the nonce  $N$  shall be set equal to the contents of the nonce field; the ciphertext  $C$  shall be set equal to the contents of the ciphertext field; and the ciphertext length  $Clen$  shall be set equal to eight times the encoded length of the ciphertext field.

## 6. Data structures

### 6.1 Presentation and encoding

Data structures in this standard are defined using Abstract Syntax Notation 1 (ASN.1) as defined in ITU-T X.680.

The data structures defined in this clause shall be encoded using the Canonical Octet Encoding Rules (COER) as defined in ITU-T X.696.

There are some data structures in this standard for which a “canonical encoding” is defined. This is the encoding to be used whenever the structures are to be encoded for processing by a cryptographic hash function. In general, these are structures that include the output of some cryptographic operation, for which the generator of the structure may choose either to include additional information to speed up receive-side processing, or to omit that additional information and reduce the transmitted packet size. The structures for which a canonical encoding is defined are HeaderInfo, HashedId8, and CertificateBase. Any structure for which encoding is subject to canonicalization has a paragraph entitled **Encoding considerations** in its description in Clause 6.

Clause 6 specifies and describes the data structures one at a time. The complete IEEE 1609.2 ASN.1 modules are given in Annex B.

### 6.2 Integer types

The following atomic types are used in the data structure definitions:

```

Uint3 ::= INTEGER (0..7)          -- (hex)      07
Uint8 ::= INTEGER (0..255)         -- (hex)      ff
Uint16 ::= INTEGER (0..65535)       -- (hex)      ff ff
Uint32 ::= INTEGER (0..4294967295)  -- (hex)      ff ff ff ff
Uint64 ::= INTEGER (0..18446744073709551615)  -- (hex)      ff ff ff ff ff ff ff ff
IValue ::= Uint16

```

The following synonym for OCTET STRING is used in the data structure definitions:

```
Opaque ::= OCTET STRING
```

The following structures are used for clarity of definitions:

```

SequenceOfOctetString ::= SEQUENCE (SIZE (0..MAX) ) OF
    OCTET STRING (SIZE(0..MAX))
SequenceOfUint3 ::= SEQUENCE OF Uint3
SequenceOfUint8 ::= SEQUENCE OF Uint8
SequenceOfUint16 ::= SEQUENCE OF Uint16

```

## 6.3 Secured protocol data units (SPDUs)

### 6.3.1 General

Subclause 6.3 specifies the secured protocol data unit (SPDU) structures created and consumed by the SDS. A SPDU is an Ieee1609Dot2Data as defined in 6.3.

The order in which the structures are defined below is hierarchical based on the first use in a prior structure. For example, in 6.2.2 Ieee1609Dot2Data is defined using several structures, of which the first three are Opaque (a synonym for OCTET STRING, see 6.2), SignedData, and EncryptedData. Subsequently, SignedData is defined in 6.3.4, and EncryptedData is defined in 6.3.30. The subclauses between 6.3.4 and 6.3.30 are used to define structures used within SignedData, and so on. (Exceptions are the fields associated with MissingCrlIdentifier, which are defined in 7.3 in order to keep all CRL-related fields in one place).

Additionally, in the electronic version of the standard, all uses of a structure name are hyperlinked to the title of the subclause that defines the structure.

### 6.3.2 Ieee1609Dot2Data

```
Ieee1609Dot2Data ::= SEQUENCE {
    protocolVersion    Uint8(3),
    content            Ieee1609Dot2Content
}
```

This data type is used to contain the other data types in this clause. The fields in the Ieee1609Dot2Data have the following meanings:

- `protocolVersion` contains the current version of the protocol. The version specified in this document is version 3, represented by the integer 3. There are no major or minor version numbers.
- `content` contains the content in the form of an Ieee1609Dot2Content.

### 6.3.3 Ieee1609Dot2Content

```
Ieee1609Dot2Content ::= CHOICE {
    unsecuredData        Opaque,
    signedData          SignedData,
    encryptedData       EncryptedData,
    signedCertificateRequest Opaque,
    ...
}
```

In this structure:

- `unsecuredData` indicates that the content is an OCTET STRING to be consumed outside the SDS.
- `signedData` indicates that the content has been signed according to this standard.
- `encryptedData` indicates that the content has been encrypted according to this standard.

- `signedCertificateRequest` indicates that the content is a certificate request. Further specification of certificate requests is not provided in this version of this standard.

#### 6.3.4 SignedData

```
SignedData ::= SEQUENCE {
    hashId          HashAlgorithm,
    tbsData         ToBeSignedData,
    signer          SignerIdentifier,
    signature       Signature
}
```

In this structure:

- `hashId` indicates the hash algorithm to be used to generate the hash of the message for signing and verification.
- `tbsData` contains the data that is hashed as input to the signature.
- `signer` determines the keying material and hash algorithm used to sign the data.
- `signature` contains the digital signature itself, calculated as specified in 5.3.1, with:
  - *Data input* equal to the COER encoding of the `tbsData` field canonicalized according to the encoding considerations given in 6.3.6.
  - *Verification type* equal to *certificate*.
  - *Signer identifier input* equal to the COER-encoding of the *Certificate* that is to be used to verify the SPDU, canonicalized according to the encoding considerations given in 6.4.3.

#### 6.3.5 HashAlgorithm

```
HashAlgorithm ::= ENUMERATED {
    sha256,
    ...
}
```

This structure identifies a hash algorithm. The only value currently supported is `sha256`, indicating SHA-256 as specified in 5.3.3.

**Critical information fields:** This is a critical information field as defined in 5.2.5. An implementation that does not recognize the enumerated value of this type in a signed SPDU when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

#### 6.3.6 ToBeSignedData

```
ToBeSignedData ::= SEQUENCE {
    payload        SignedDataPayload,
    headerInfo     HeaderInfo
}
```

This structure contains the data to be hashed when generating or verifying a signature. See 6.3.4 for the specification of the input to the hash.

- payload contains data that is provided by the entity that invokes the SDS.
- headerInfo contains additional data that is inserted by the SDS.

**Encoding considerations:** For encoding considerations associated with the headerInfo field, see 6.3.9.

### 6.3.7 SignedDataPayload

```
SignedDataPayload ::= SEQUENCE {
    data                  Ieee1609Dot2Data OPTIONAL,
    extDataHash          HashedData OPTIONAL,
    ...
}
(WITH COMPONENTS {...., data PRESENT} |
 WITH COMPONENTS {...., extDataHash PRESENT})
```

This structure contains the data payload of a ToBeSignedData. This structure contains at least one of data and extDataHash, and may contain both.

- data contains data that is explicitly transported within the structure.
- extDataHash contains the hash of data that is not explicitly transported within the structure, and which the creator of the structure wishes to cryptographically bind to the signature. For example, if a creator wanted to indicate that some large message was still valid, they could use the extDataHash field to send a SignedData containing the hash of that large message without having to resend the message itself. Whether or not extDataHash is used, and how it is used, is SDEE-specific.

### 6.3.8 HashedData

```
HashedData ::= CHOICE {
    sha256HashedData OCTET STRING (SIZE(32)),
    ...
}
```

This structure contains the hash of some data with a specified hash algorithm. The only hash algorithm supported in this version of this standard is SHA-256.

**Critical information fields:** If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE for this type when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

### 6.3.9 HeaderInfo

```
HeaderInfo ::= SEQUENCE {
    psid                  Psid,
    generationTime        Time64 OPTIONAL,
    expiryTime            Time64 OPTIONAL,
    generationLocation   ThreeDLocation OPTIONAL,
```

```

p2pcdLearningRequest      HashedId3 OPTIONAL,
missingCrlIdentifier     MissingCrlIdentifier OPTIONAL,
encryptionKey            EncryptionKey OPTIONAL,
...
}

```

This structure contains information that is used to establish validity by the criteria of 5.2.

- `psid` indicates the application area with which the sender is claiming the payload should be associated.
- `generationTime` indicates the time at which the structure was generated. See 5.2.4.2.2 and 5.2.4.2.3 for discussion of the use of this field.
- `expiryTime`, if present, contains the time after which the data should no longer be considered relevant. If both `generationTime` and `expiryTime` are present, the signed SPDU is invalid if `generationTime` is not strictly earlier than `expiryTime`.
- `generationLocation`, if present, contains the location at which the signature was generated.
- `p2pcdLearningRequest`, if present, is used by the SDS to request certificates for which it has seen identifiers but do not know the entire certificate. A specification of this peer-to-peer certificate distribution mechanism is given in Clause 8.
- `missingCrlIdentifier`, if present, is used by the SDS to request CRLs which it knows to have been issued but have not received. This is provided for future use and the associated mechanism is not defined in this version of this standard.
- `encryptionKey`, if present, is used to indicate that a further communication should be encrypted with the indicated key. One possible use of this key to encrypt a response is specified in 6.3.1, 6.3.3, and 6.3.31. An `encryptionKey` field of type symmetric should only be used if the Signed-Data containing this field is securely encrypted by some means.

**Encoding considerations:** When the structure is encoded in order to be digested to generate or check a signature, if `encryptionKey` is present, and indicates the choice public, and contains a `BasePublicEncryptionKey` that is an elliptic curve point (i.e., of type `EccP256CurvePoint`), then the elliptic curve point is encoded in compressed form, i.e., such that the choice indicated within the `EccP256CurvePoint` is compressed-y-0 or compressed-y-1.

### 6.3.10 Psid

```
Psid ::= INTEGER (0..MAX)
```

This type represents the PSID defined in IEEE Std 1609.12.

### 6.3.11 Time64

```
Time64 ::= Uint64
```

This data structure is a 64-bit integer giving an estimate of the number of (TAI) microseconds since 00:00:00 UTC, 1 January, 2004.

### 6.3.12 ThreeDLocation

```
ThreeDLocation ::=  
SEQUENCE { latitude Latitude,  
           longitude Longitude,  
           elevation Elevation  
 }
```

This data structure contains an estimate of 3D location. The details of the structure are given in the definitions of the individual fields below.

NOTE—The units used in this data structure are consistent with the location data structures used in SAE J2735 [B20], though the encoding is incompatible.

### 6.3.13 Latitude

```
Latitude ::= NinetyDegreeInt  
NinetyDegreeInt ::= INTEGER {  
    min (-900000000),  
    max (900000000),  
    unknown (900000001)  
} (-900000000..900000001)  
  
KnownLatitude ::= NinetyDegreeInt (min..max)  
-- Minus 90deg to +90deg in .1 microdegree intervals  
UnknownLatitude ::= NinetyDegreeInt (unknown)
```

The `latitude` field contains an `INTEGER` encoding an estimate of the latitude with precision 1/10<sup>th</sup> microdegree relative to the World Geodetic System (WGS)-84 datum as defined in NIMA Technical Report TR8350.2.

The integer in the `latitude` field is no more than 900 000 000 and no less than -900 000 000, except that the value 900 000 001 is used to indicate the latitude was not available to the sender.

### 6.3.14 Longitude

```
Longitude ::= OneEightyDegreeInt  
OneEightyDegreeInt ::= INTEGER {  
    min (-1799999999),  
    max (1800000000),  
    unknown (1800000001)  
} (-1799999999..1800000001)  
  
KnownLongitude ::= OneEightyDegreeInt (min..max)  
UnknownLongitude ::= OneEightyDegreeInt (unknown)
```

The `longitude` field contains an `INTEGER` encoding an estimate of the longitude with precision 1/10<sup>th</sup> microdegree relative to the World Geodetic System (WGS)-84 datum as defined in NIMA Technical Report TR8350.2.

The integer in the `longitude` field is no more than 1 800 000 000 and no less than -1 799 999 999, except that the value 1 800 000 001 is used to indicate that the longitude was not available to the sender.

### 6.3.15 Elevation

```
Elevation ::= ElevInt
ElevInt ::= Uint16
```

The elevation field contains an estimate of the geodetic altitude above or below the WGS84 ellipsoid. The 16-bit value is interpreted as an integer number of decimeters representing the height above a minimum height of -409.5 m, with the maximum height being 6143.9 m.

### 6.3.16 MissingCrlIdentifier

```
MissingCrlIdentifier ::= 
    SEQUENCE { cracaId          HashedId3,
                crlSeries        CrlSeries,
                ...
            }
```

This structure may be used to request a CRL that the SSME knows to have been issued but has not yet received. It is provided for future use and its use is not defined in this version of this standard.

- `cracaId` is the `HashedId3` of the CRACA, as defined in 5.1.3.
- `crlSeries` is the requested CRL Series value. See 5.1.3 for more information.

### 6.3.17 CrlSeries

```
CrlSeries ::= Uint16
```

This integer identifies a series of CRLs issued under the authority of a particular CRACA.

### 6.3.18 EncryptionKey

```
EncryptionKey ::= CHOICE {
    public           PublicEncryptionKey,
    symmetric        SymmetricEncryptionKey
}
```

This structure contains an encryption key, which may be a public or a symmetric key.

### 6.3.19 SymmetricEncryptionKey

```
SymmetricEncryptionKey ::= CHOICE {
    aes128Ccm      OCTET STRING(SIZE(16)),
    ...
}
```

This structure provides the key bytes for use with an identified symmetric algorithm. The only supported symmetric algorithm is AES-128 in CCM mode as specified in 5.3.8.

**Critical information fields:** If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE for this type when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

### 6.3.20 PublicEncryptionKey

```
PublicEncryptionKey ::= SEQUENCE {
    supportedSymmAlg      SymmAlgorithm,
    publicKey              BasePublicEncryptionKey
}
```

This structure specifies a public encryption key and the associated symmetric algorithm which is used for bulk data encryption when encrypting for that public key.

### 6.3.21 SymmAlgorithm

```
SymmAlgorithm ::= ENUMERATED {
    aes128Ccm,
    ...
}
```

This enumerated value indicates supported symmetric algorithms. The only symmetric algorithm supported in this version of this standard is AES-CCM as specified in 5.3.8.

**Critical information fields:** If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated enumerated value for this type when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

### 6.3.22 BasePublicEncryptionKey

```
BasePublicEncryptionKey ::= CHOICE {
    eciesNistP256          EccP256CurvePoint,
    eciesBrainpoolP256r1   EccP256CurvePoint,
    ...
}
```

This structure specifies the bytes of a public encryption key for a particular algorithm. The only algorithm supported is ECIES over either the NIST P256 or the Brainpool P256r1 curve as specified in 5.3.5.

**Critical information fields:** If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE for this type when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

### 6.3.23 EccP256CurvePoint

```
EccP256CurvePoint ::= CHOICE {
    x-only           OCTET STRING (SIZE (32)),
    fill             NULL, -- consistency w 1363 / X9.62
    compressed-y-0   OCTET STRING (SIZE (32)),
    compressed-y-1   OCTET STRING (SIZE (32)),
    uncompressed     SEQUENCE {
        x OCTET STRING (SIZE (32)),
        y OCTET STRING (SIZE (32))
    }
}
```

This structure specifies a point on an elliptic curve in Weierstrass form defined over a 256-bit prime number. This encompasses both NIST p256 as defined in FIPS 186-4 and Brainpool p256r1 as defined in RFC 5639. The fields in this structure are OCTET STRINGS produced with the elliptic curve point encoding and decoding methods defined in subclause 5.5.6 of IEEE Std 1363-2000. The x-coordinate is encoded as an unsigned integer of length 32 octets in network byte order for all values of the CHOICE; the encoding of the y-coordinate y depends on whether the point is x-only, compressed, or uncompressed. If the point is x-only, y is omitted. If the point is compressed, the value of type depends on the LSB of y: if the LSB of y is 0, type takes the value compressed-y-0, and if the LSB of y is 1, type takes the value compressed-y-1. If the point is uncompressed, y is encoded explicitly as an unsigned integer of length 32 octets in network byte order.

### 6.3.24 SignerIdentifier

```
SignerIdentifier ::= CHOICE      {  
    digest          HashedId8,  
    certificate     SequenceOfCertificate,  
    self            NULL,  
    ...  
}
```

This structure allows the recipient of data to determine which keying material to use to authenticate the data. It also indicates the verification type to be used to generate the hash for verification, as specified in 5.3.1.

- If the choice indicated is `digest`:
  - The structure contains the HashedId8 of the relevant certificate, obtained as specified in the description of the HashedId8 structure.
  - The verification type is *certificate*.
- If the choice indicated is `certificate`:
  - The structure contains one or more Certificate structures, in order such that the first certificate is the authorization certificate and each subsequent certificate is the issuer of the one before it.
  - The verification type is *certificate* and the certificate data passed to the hash function as specified in 5.3.1 is the authorization certificate.
- If the choice indicated is `self`:
  - The structure does not contain any data beyond the indication that the choice value is `self`.
  - The verification type is *self-signed*.

#### Critical information fields:

- a) If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the CHOICE value for this type when verifying a signed SPDU shall indicate that the signed SPDU is invalid.
- b) If present, `certificate` is a critical information field as defined in 5.2.5. An implementation that does not support the number of certificates in `certificate` when verifying a signed SPDU shall indicate that the signed SPDU is invalid. A compliant implementation shall support `certificate` fields containing at least one certificate.

### 6.3.25 HashedId3

```
HashedId3 ::= OCTET STRING (SIZE(3))
```

This data structure contains the truncated hash of another data structure. The HashedId3 for a given data structure is calculated by calculating the SHA-256 hash of the encoded data structure and taking the low-order three bytes of the hash output. The low-order three bytes are the last three bytes of the 32-byte hash.

**Encoding considerations:** If the data structure is a Certificate, the encoded Certificate which is input to the hash uses the compressed form for all elliptic curve points within the ToBeSignedCertificate and takes the *r* value of an ECDSA signature to be of type x-only. If the data structure is an Ieee1609Dot2Data containing a SignedData, the encoding takes the *r* value of an ECDSA signature to be of type x-only.

### 6.3.26 HashedId8

```
HashedId8 ::= OCTET STRING (SIZE(8))
```

This data structure contains the truncated hash of another data structure. The HashedId8 for a given data structure is calculated by calculating the SHA-256 hash of the encoded data structure and taking the low-order eight bytes of the hash output. The low-order eight bytes are the last eight bytes of the 32-byte hash.

**Encoding considerations:** If the data structure is a Certificate, the encoded Certificate which is input to the hash uses the compressed form for all elliptic curve points within the ToBeSignedCertificate and shall take the *r* value of an ECDSA signature to be of type x-only. If the data structure is an Ieee1609Dot2Data containing a SignedData, the encoding takes the *r* value of an ECDSA signature to be of type x-only.

### 6.3.27 HashedId10

```
HashedId10 ::= OCTET STRING (SIZE(10))
```

This data structure contains the truncated hash of another data structure. The HashedId10 for a given data structure is calculated by calculating the SHA-256 hash of the encoded data structure and taking the low-order ten bytes of the hash output. The low-order ten bytes are the last ten bytes of the 32-byte hash.

**Encoding considerations:** If the data structure is a Certificate, the encoded Certificate which is input to the hash uses the compressed form for all elliptic curve points within the ToBeSignedCertificate and takes the *r* value of an ECDSA signature to be of type x-only. If the data structure is an Ieee1609Dot2Data containing a SignedData, the encoding takes the *r* value of an ECDSA signature to be of type x-only.

### 6.3.28 Signature

```
Signature ::=  
CHOICE { ecdsaNistP256Signature EcdsaP256Signature,  
ecdsaBrainpoolP256r1Signature EcdsaP256Signature,  
...  
}
```

This structure represents a signature for a supported public key algorithm. It may be contained within SignedData or Certificate.

### 6.3.29 EcDSA P256Signature

```
EcDSA P256Signature ::=  
SEQUENCE { r EccP256CurvePoint,  
           s OCTET STRING (SIZE (32))  
 }
```

This structure represents an ECDSA signature. The signature is generated as specified in 5.3.1.

If the signature process followed the specification of FIPS 186-4 and output the integer  $r$ ,  $r$  is represented as an EccP256CurvePoint indicating the selection x-only.

If the signature process followed the specification of SEC 1 and output the elliptic curve point  $R$  to allow for fast verification,  $R$  is represented as an EccP256CurvePoint indicating the choice compressed-y-0, compressed-y-1, or uncompressed at the sender's discretion.<sup>11</sup>

### 6.3.30 EncryptedData

```
EncryptedData ::=  
SEQUENCE { recipients SequenceOfRecipientInfo,  
           ciphertext SymmetricCiphertext  
 }
```

This data structure encodes data that has been encrypted to one or more recipients using the recipients' public or symmetric keys as specified in 5.3.4.

The type contains the following fields:

- recipients contains one or more RecipientInfos, defined below. If the ciphertext was produced using the static data encryption key approach specified in 5.3.4.2, recipients contains a single entry of type PreSharedKeyRecipientInfo. If the ciphertext was produced using the ephemeral data encryption key approach specified in 5.3.4.1, recipients contains one or more entries which are of any type other than PreSharedKeyRecipientInfo.
- ciphertext contains the encrypted data. This is the encryption of an encoded IEEE1609Dot2Data structure.

#### Critical information fields:

- If present, recipients is a critical information field as defined in 5.2.5. An implementation that does not support the number of RecipientInfo in recipients when decrypted shall indicate that the encrypted SPDU could not be decrypted due to unsupported critical information fields. A compliant implementation shall support recipients fields containing at least eight entries.

### 6.3.31 RecipientInfo

```
RecipientInfo ::=  
CHOICE { pskRecipInfo PreSharedKeyRecipientInfo,  
         symmRecipInfo SymmRecipientInfo,  
         certRecipInfo PKRecipientInfo,  
         signedDataRecipInfo PKRecipientInfo,
```

---

<sup>11</sup> The compressed forms give some performance advantage on verification compared to the x-only form, at the same packet size as the x-only form; the uncompressed form gives a greater performance advantage at the cost of increased packet size.

```
rekRecipInfo          PKRecipientInfo
}

```

```
SequenceOfRecipientInfo ::= SEQUENCE OF RecipientInfo
```

This data structure is used to transfer the data encryption key to an individual recipient of an EncryptedData. The option pskRecipInfo is selected if the EncryptedData was encrypted using the static encryption key approach specified in 5.3.4.2. The other options are selected if the EncryptedData was encrypted using the ephemeral encryption key approach specified in 5.3.4.1. The meanings of the choices are:

- pskRecipInfo: The ciphertext was encrypted directly using a symmetric key.
- symmRecipInfo: The data encryption key was encrypted using a symmetric key.
- certRecipInfo: The data encryption key was encrypted using the public encryption key in a certificate. This field contains the HashedId8 of the certificate. In this case, the parameter P1 to ECIES as defined in 5.3.5 is the hash of the certificate.
- signedDataRecipInfo: The data encryption key was encrypted using the public response encryption key from a SignedData. In this case, this field contains the HashedId8 of the Ieee1609Dot2Data containing the SignedData containing the encryption key. In this case, the parameter P1 to ECIES as defined in 5.3.5 is the SHA-256 hash of the Ieee1609Dot2Data containing the response encryption key.
- rekRecipInfo: The data encryption key was encrypted using a public response encryption key that was not obtained from a SignedData. In this case, this field contains the HashedId8 of the response encryption key. In this case, the parameter P1 to ECIES as defined in 5.3.5 is the hash of the empty string.

NOTE—The rekRecipInfo should only be used if the SignedData is not available as it potentially allows misbinding attacks: it is included in these structures specifically to enable certificate response encryption from a PCA to an end-entity device.

### 6.3.1 PreSharedKeyRecipientInfo

```
PreSharedKeyRecipientInfo ::= HashedId8
```

This data structure is used to indicate a symmetric key that may be used directly to decrypt a SymmetricCiphertext. It consists of the low-order 8 bytes of the SHA-256 hash of the symmetric key in question. The symmetric key may be established by any appropriate means agreed by the two parties to the exchange.

### 6.3.2 SymmRecipientInfo

```
SymmRecipientInfo ::= SEQUENCE {
    recipientId      HashedId8,
    encKey           SymmetricCiphertext
}
```

This data structure contains the following fields:

- recipientId contains the hash of the symmetric key encryption key that may be used to decrypt the data encryption key.
- encKey contains the encrypted data encryption key within an AES-CCM ciphertext.

### 6.3.3 PKRecipientInfo

```
PKRecipientInfo ::= SEQUENCE {
    recipientId      HashedId8,
    encKey           EncryptedDataEncryptionKey
}
```

This data structure contains the following fields:

- `recipientId` contains the hash of the container for the encryption public key as specified in the definition of `RecipientInfo`.
- `encKey` contains the encrypted key.

### 6.3.4 EncryptedDataEncryptionKey

```
EncryptedDataEncryptionKey ::= CHOICE {
    eciesNistP256          EciesP256EncryptedKey,
    eciesBrainpoolP256r1    EciesP256EncryptedKey,
    ...
}
```

This data structure contains an encrypted data encryption key.

**Critical information fields:** If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated enumerated value for this type in an encrypted SPDU shall reject the SPDU as invalid.

### 6.3.5 EciesP256EncryptedKey

```
EciesP256EncryptedKey :=
    SEQUENCE { v           EccP256CurvePoint,
                c           OCTET STRING (SIZE (16)),
                t           OCTET STRING (SIZE (16))
    }
```

This data structure is used to transfer a 16-byte symmetric key encrypted using ECIES as specified in IEEE Std 1363a-2004. The type contains the following fields:

- `v` is the sender's ephemeral public key, which is the output  $V$  from encryption as specified in 5.3.5.
- `c` is the encrypted symmetric key, which is the output  $C$  from encryption as specified in 5.3.5. The algorithm for the symmetric key is identified by the CHOICE indicated in the following `SymmetricCiphertext`.
- `t` is the authentication tag, which is the output `tag` from encryption as specified in 5.3.5.

Encryption and decryption are carried out as specified in 5.3.5.

### 6.3.6 SymmetricCiphertext

```
SymmetricCiphertext :=
    CHOICE { aes128ccm        AesCcmCiphertext,
            ...
    }
```

}

This data structure encapsulates a ciphertext generated with an approved symmetric algorithm.

**Critical information fields:** If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE value for this type in an encrypted SPDU shall reject the SPDU as invalid.

### 6.3.7 AesCcmCiphertext

```
AesCcmCiphertext ::=  
SEQUENCE { nonce OCTET STRING (SIZE (12)),  
          ccmCiphertext Opaque  
}
```

This data structure encapsulates an encrypted ciphertext for the AES-CCM symmetric algorithm. It contains the following fields:

- nonce contains the nonce  $N$  as specified in 5.3.8.
- ccmCiphertext contains the ciphertext  $C$  as specified in 5.3.8.

The ciphertext is 16 bytes longer than the corresponding plaintext.

The plaintext resulting from a correct decryption of the ciphertext is a COER-encoded Ieee1609Dot2Data structure.

## 6.4 Certificates and other security management data structures

### 6.4.1 General

Subclause 6.4 specifies the structures to be used for certificates and security management.

### 6.4.2 Certificate

```
Certificate ::=  
CertificateBase (ImplicitCertificate | ExplicitCertificate)  
  
SequenceOfCertificate ::= SEQUENCE OF Certificate
```

This structure is a profile of the structure CertificateBase which specifies the valid combinations of fields to transmit implicit and explicit certificates.

### 6.4.3 CertificateBase

```
CertificateBase ::=  
SEQUENCE { version Uint8 (3),  
          type CertificateType,  
          issuer IssuerIdentifier,  
          toBeSigned ToBeSignedCertificate,
```

signature	Signature OPTIONAL
}	

The fields in this structure have the following meaning:

- `version` contains the version of the certificate format. In this version of the data structures, this field is set to 3.
- `type` states whether the certificate is implicit or explicit. This field is set to `explicit` for explicit certificates and to `implicit` for implicit certificates. See `ExplicitCertificate` and `ImplicitCertificate` for more details.
- `issuer` identifies the issuer of the certificate.
- `toBeSigned` is the certificate contents. This field is an input to the hash when generating or verifying signatures for an explicit certificate, or generating or verifying the public key from the reconstruction value for an implicit certificate. The details of how this field are encoded are given in the description of the `ToBeSignedCertificate` `type`.
- `signature` is included in an `ExplicitCertificate`. It is the signature, calculated by the signer identified in the `issuer` field, over the hash of `toBeSigned`. The hash is calculated as specified in 5.3.1, where:
  - *Data input* is the encoding of `toBeSigned` following the COER.
  - *Signer identifier input* depends on the verification type, which in turn depends on the choice indicated by `issuer`. If the choice indicated by `issuer` is `self`, the verification type is `self-signed` and the *signer identifier input* is the empty string. If the choice indicated by `issuer` is not `self`, the verification type is `certificate` and the *signer identifier input* is the canonicalized COER encoding of the certificate indicated by `issuer`. The canonicalization is carried out as specified in the **Encoding considerations** section of this subclause.

**Encoding considerations:** When a certificate is encoded for hashing, for example to generate its `HashedId8`, or when it is to be used as the *signer identifier information* for verification, it is canonicalized as follows:

- The encoding of `toBeSigned` uses the compressed form for all elliptic curve points: that is, those points (which in this standard are all `EccP256CurvePoints`) indicate a choice of `compressed-y-0` or `compressed-y-1`.
- The encoding of the signature, if present and if an ECDSA signature, takes the `r` value to be an `EccP256CurvePoint` indicating the choice `x-only`.

#### 6.4.4 CertificateType

```
CertificateType ::= ENUMERATED {
    explicit,
    implicit,
    ...
}
```

This enumerated type indicates whether a certificate is explicit or implicit.

#### 6.4.5 ImplicitCertificate

```
ImplicitCertificate ::= CertificateBase (WITH COMPONENTS { . . . ,
    type(implicit),
    toBeSigned(WITH COMPONENTS { . . . ,
        verifyKeyIndicator(WITH COMPONENTS {reconstructionValue})
    }),
    signature ABSENT
} )
```

This is a profile of the CertificateBase structure providing all the fields necessary for an implicit certificate, and no others.

#### 6.4.6 ExplicitCertificate

```
ExplicitCertificate ::= CertificateBase (WITH COMPONENTS { . . . ,
    type(explicit),
    toBeSigned(WITH COMPONENTS { . . . ,
        verifyKeyIndicator(WITH COMPONENTS {verificationKey})
    }),
    signature PRESENT
} )
```

This is a profile of the CertificateBase structure providing all the fields necessary for an explicit certificate, and no others.

#### 6.4.7 IssuerIdentifier

```
IssuerIdentifier ::= CHOICE {
    sha256AndDigest      HashedId8,
    self                  HashAlgorithm,
    ...
}
```

This structure allows the recipient of a certificate to determine which keying material to use to authenticate the certificate.

If the choice indicated is `sha256AndDigest`:

- The structure contains the `HashedId8` of the issuing certificate, obtained as specified in the description of the `HashedId8` structure.
- The hash algorithm to be used to generate the hash of the certificate for verification is SHA-256.
- The certificate is to be verified with the public key of the indicated issuing certificate.

If the choice indicated is `self`:

- The structure indicates what hash algorithm is to be used to generate the hash of the certificate for verification.
- The certificate is to be verified with the public key indicated by the `verifyKeyIndicator` field in the `toBeSignedCertificate`.

#### 6.4.8 ToBeSignedCertificate

```

ToBeSignedCertificate ::= SEQUENCE {
    id                                CertificateId,
    cracaId                            HashedId3,
    crlSeries                           CrlSeries,
    validityPeriod                     ValidityPeriod,
    region                             GeographicRegion OPTIONAL,
    assuranceLevel                     SubjectAssurance OPTIONAL,
    appPermissions                      SequenceOfPsidSsp OPTIONAL,
    certIssuePermissions                SequenceOfPsidGroupPermissions OPTIONAL,
    certRequestPermissions              SequenceOfPsidGroupPermissions OPTIONAL,
    canRequestRollover                 NULL OPTIONAL,
    encryptionKey                      PublicEncryptionKey OPTIONAL,
    verifyKeyIndicator                 VerificationKeyIndicator,
    ...
}
(WITH COMPONENTS { ..., appPermissions PRESENT} |
 WITH COMPONENTS { ..., certIssuePermissions PRESENT} |
 WITH COMPONENTS { ..., certRequestPermissions PRESENT})

```

The fields in the ToBeSignedCertificate structure have the following meaning:

- `id` contains information that is used to identify the certificate holder if necessary.
- `cracaId` identifies the Certificate Revocation Authorization CA (CRACA) responsible for certificate revocation lists (CRLs) on which this certificate might appear. Use of the `cracaId` is specified in 5.1.3.
- `crlSeries` represents the CRL series relevant to a particular Certificate Revocation Authorization CA (CRACA) on which the certificate might appear. Use of this field is specified in 5.1.3.
- `validityPeriod` contains the validity period of the certificate.
- `region`, if present, indicates the validity region of the certificate. If it is omitted the validity region is indicated as follows:
  - If enclosing certificate is self-signed, i.e., the choice indicated by the `issuer` field in the enclosing certificate structure is self, the certificate is valid worldwide.
  - Otherwise, the certificate has the same validity region as the certificate that issued it.
- `assuranceLevel` indicates the assurance level of the certificate holder.
- `appPermissions` indicates the permissions that the certificate holder has to sign application data with this certificate. A valid instance of `appPermissions` contains any particular Psid value in at most one entry.
- `certIssuePermissions` indicates the permissions that the certificate holder has to sign certificates with this certificate. A valid instance of this array contains no more than one entry whose `psidSspRange` field indicates all. If the array has multiple entries and one entry has its `psidSspRange` field indicate all, then the entry indicating all specifies the permissions for all PSIDs other than the ones explicitly specified in the other entries. See the description of `PsidGroupPermissions` for further discussion.
- `certRequestPermissions` indicates the permissions that the certificate holder has to sign certificate requests with this certificate. A valid instance of this array contains no more than one entry whose `psidSspRange` field indicates all. If the array has multiple entries and one entry has its

psidSspRange field indicate all, then the entry indicating all specifies the permissions for all PSIDs other than the ones explicitly specified in the other entries. See the description of PsidGroupPermissions for further discussion.

- canRequestRollover indicates that the certificate may be used to sign a request for another certificate with the same permissions. This field is provided for future use and its use is not defined in this version of this standard.
- encryptionKey contains a public key for encryption for which the certificate holder holds the corresponding private key.
- verifyKeyIndicator contains material that may be used to recover the public key that may be used to verify data signed by this certificate.

**Encoding considerations:** The encoding of `toBeSigned` which is input to the hash uses the compressed form for all public keys and reconstruction values that are elliptic curve points: that is, those points (which in this standard are all `EccP256CurvePoints`) indicate a choice of `compressed-y=0` or `compressed-y=1`. The encoding of the issuing certificate uses the compressed form for all public key and reconstruction values and takes the *r* value of an ECDSA signature, which in this standard is an `EccP256CurvePoint`, to be of type `x-only`.

For both implicit and explicit certificates, when the certificate is hashed to create or recover the public key (in the case of an implicit certificate) or to generate or verify the signature (in the case of an explicit certificate), the hash is  $\text{Hash}(\text{Data input}) \parallel \text{Hash}(\text{Signer identifier input})$ , where:

- *Data input* is the COER encoding of `toBeSigned`, canonicalized as described above.
- *Signer identifier input* depends on the verification type, which in turn depends on the choice indicated by `issuer`. If the choice indicated by `issuer` is `self`, the verification type is `self-signed` and the *signer identifier input* is the empty string. If the choice indicated by `issuer` is not `self`, the verification type is `certificate` and the *signer identifier input* is the COER encoding of the canonicalization per 6.4.3 of the certificate indicated by `issuer`.

In other words, for implicit certificates, the value H (CertU) in SEC 4, section 3, is for purposes of this standard taken to be H [H (canonicalized ToBeSignedCertificate from the subordinate certificate) || H (entirety of issuer Certificate)]. See 5.3.2 for further discussion.

NOTE—This encoding of the implicit certificate for hashing has been changed from the encoding specified in IEEE Std 1609.2-2013 for consistency with the encoding of the explicit certificates. This definition of the encoding results in implicit and explicit certificates both being hashed as specified in 5.3.1.

#### Critical information fields:

- If present, `appPermissions` is a critical information field as defined in 5.2.5. An implementation that does not support the number of `PsidSsp` in `appPermissions` shall reject the encrypted SPDU as invalid. A compliant implementation shall support `appPermissions` fields containing at least eight entries.
- If present, `certIssuePermissions` is a critical information field as defined in 5.2.5. An implementation that does not support the number of `PsidGroupPermissions` in `certIssuePermissions` shall reject the encrypted SPDU as invalid. A compliant implementation shall support `certIssuePermissions` fields containing at least eight entries.
- If present, `certRequestPermissions` is a critical information field as defined in 5.2.5. An implementation that does not support the number of `PsidGroupPermissions` in `certRequestPermissions` shall reject the encrypted SPDU as invalid. A compliant

implementation shall support certRequestPermissions fields containing at least eight entries.

#### 6.4.9 CertificateId

```
CertificateId ::= CHOICE {
    linkageData           LinkageData,
    name                  Hostname,
    binaryId              OCTET STRING(SIZE(1..64)),
    none                 NULL,
    ...
}
```

This structure contains information that is used to identify the certificate holder if necessary.

- linkageData is used to identify the certificate for revocation purposes in the case of certificates that appear on linked certificate CRLs. See 5.1.3 and 7.3 for further discussion.
- name is used to identify the certificate holder in the case of non-anonymous certificates. The contents of this field are a matter of policy but should be human-readable.
- binaryId supports identifiers that are not human-readable.
- none indicates that the certificate does not include an identifier.

#### Critical information fields:

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the choice indicated in this field shall reject a signed SPDU as invalid.

#### 6.4.10 LinkageData

```
LinkageData ::= 
    SEQUENCE { iCert          IValue,
                linkage-value   LinkageValue,
                group-linkage-value GroupLinkageValue OPTIONAL
            }
```

This structure contains information that is matched against information obtained from a linkage ID-based CRL to determine whether the containing certificate has been revoked. See 5.1.3.4 and 7.3 for details of use.

#### 6.4.11 LinkageValue

```
LinkageValue ::= OCTET STRING (SIZE(9))
```

This is the individual linkage value. See 5.1.3 and 7.3 for details of use.

#### 6.4.12 GroupLinkageValue

```
GroupLinkageValue ::= SEQUENCE {
    jValue      OCTET STRING (SIZE(4)),
```

```
    value    OCTET STRING (SIZE(9))  
}
```

This is the group linkage value. See 5.1.3 and 7.3 for details of use.

#### 6.4.13 Hostname

```
Hostname ::= UTF8String (SIZE(0..255))
```

This is a UTF-8 string as defined in IETF RFC 3629. The contents are determined by policy.

#### 6.4.14 ValidityPeriod

```
ValidityPeriod ::= SEQUENCE {  
    start          Time32,  
    duration       Duration  
}
```

This type gives the validity period of a certificate. The start of the validity period is given by `start` and the end is given by `start + duration`.

#### 6.4.15 Time32

```
Time32 ::= Uint32
```

This type gives the number of (TAI) seconds since 00:00:00 UTC, 1 January, 2004.

#### 6.4.16 Duration

```
Duration ::= CHOICE {  
    microseconds   Uint16,  
    milliseconds  Uint16,  
    seconds        Uint16,  
    minutes        Uint16,  
    hours          Uint16,  
    sixtyHours    Uint16,  
    years          Uint16  
}
```

This type represents the duration of validity of a certificate. The `Uint16` value is the duration, given in the units denoted by the indicated choice. A year is considered to be 31556952 seconds, which is the average number of seconds in a year; if it is desired to map years more closely to wall-clock days, this can be done using the `hours` choice for up to seven years and the `sixtyHours` choice for up to 448.

#### 6.4.17 GeographicRegion

```
GeographicRegion ::=  
    CHOICE { circularRegion      CircularRegion,  
             rectangularRegion   SequenceOfRectangularRegion,  
             polygonalRegion     PolygonalRegion,
```

```
identifiedRegion    SequenceOfIdentifiedRegion,
...
}
```

This type represents a geographic region of a specified form.

- `rectangularRegion` is an array of `RectangularRegion` structures containing at least one entry. This field is interpreted as a series of rectangles, which may overlap or be disjoint. The permitted region is any point within any of the rectangles.
- `circularRegion` or `polygonalRegion` contain a single instance of their respective types.
- `identifiedRegion` is an array of `IdentifiedRegion` structures containing at least one entry. The permitted region is any point within any of the identified regions.

A certificate is not valid if any part of the region indicated in its scope field lies outside the region indicated in the scope of its issuer.

#### **Critical information fields:**

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE when verifying a signed SPDU shall indicate that the signed SPDU is invalid.
- If selected, `rectangularRegion` is a critical information field as defined in 5.2.5. An implementation that does not support the number of `RectangularRegion` in `rectangularRegions` when verifying a signed SPDU shall indicate that the signed SPDU is invalid. A compliant implementation shall support `rectangularRegions` fields containing at least eight entries.
- If selected, `identifiedRegion` is a critical information field as defined in 5.2.5. An implementation that does not support the number of `IdentifiedRegion` in `identifiedRegion` shall reject the encrypted SPDU as invalid. A compliant implementation shall support `identifiedRegion` fields containing at least eight entries.

#### **6.4.18 CircularRegion**

```
CircularRegion ::=

SEQUENCE { center           TwoDLocation,
            radius            Uint16
        }
```

This structure specifies a circle with its center at `center`, its `radius` given in meters, and located tangential to the reference ellipsoid. The indicated region is all the points on the surface of the reference ellipsoid whose distance to the center point over the reference ellipsoid is less than or equal to the radius. A point which contains an elevation component is considered to be within the circular region if its horizontal projection onto the reference ellipsoid lies within the region.

#### **6.4.19 TwoDLocation**

```
TwoDLocation ::=

SEQUENCE { latitude       Latitude,
            longitude      Longitude
        }
```

This data structure is used to define validity regions for use in certificates. The *latitude* and *longitude* fields contain the latitude and longitude as defined above.

NOTE—This data structure is consistent with the location encoding used in SAE J2735 [B20], except that values 900 000 001 for latitude (used to indicate that the latitude was not available) and 1 800 000 001 for longitude (used to indicate that the longitude was not available) are not valid.

#### 6.4.20 RectangularRegion

```
RectangularRegion ::=  
SEQUENCE { northWest      TwoDLocation,  
           southEast       TwoDLocation  
 }
```

```
SequenceOfRectangularRegion ::= SEQUENCE OF RectangularRegion
```

This structure specifies a rectangle formed by connecting in sequence: (northWest.latitude, northWest.longitude), (southEast.latitude, northWest.longitude), (southEast.latitude, southEast.longitude), and (northWest.latitude, southEast.longitude). The points are connected by lines of constant latitude or longitude. A point which contains an elevation component is considered to be within the rectangular region if its horizontal projection onto the reference ellipsoid lies within the region. A RectangularRegion is valid only if the northWest value is north and west of the southEast value, i.e., the two points cannot have equal latitude or equal longitude.

#### 6.4.21 PolygonalRegion

```
PolygonalRegion ::= SEQUENCE SIZE(3..MAX) OF TwoDLocation
```

This data structure defines a region using a series of distinct geographic points, defined on the surface of the reference ellipsoid. The region is specified by connecting the points in the order they appear, with each pair of points connected by the geodesic on the reference ellipsoid. The polygon is completed by connecting the final point to the first point. The allowed region is the interior of the polygon and its boundary.

A point which contains an elevation component is considered to be within the polygonal region if its horizontal projection onto the reference ellipsoid lies within the region.

A valid PolygonalRegion contains at least three points. In a valid PolygonalRegion, the implied lines that make up the sides of the polygon do not intersect.

#### Critical information fields:

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not support the number of TwoDLocation in the PolygonalRegion when verifying a signed SPDUs shall indicate that the signed SPDUs is invalid. A compliant implementation shall support PolygonalRegions containing at least eight TwoDLocation entries.

#### 6.4.22 IdentifiedRegion

```
IdentifiedRegion ::=  
CHOICE { countryOnly      CountryOnly,
```

```
countryAndRegions      CountryAndRegions,
countryAndSubregions   CountryAndSubregions,
...
}

SequenceOfIdentifiedRegion ::= SEQUENCE OF IdentifiedRegion
```

This type indicates the region of validity of a certificate using region identifiers.

#### Critical information fields:

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

#### 6.4.23 CountryOnly

```
CountryOnly ::= Uint16
```

This is the integer representation of the country or area identifier as defined by the United Nations Statistics Division in October 2013 (see normative references in Clause 2).

#### 6.4.24 CountryAndRegions

```
CountryAndRegions ::= SEQUENCE {
    countryOnly      CountryOnly,
    regions          SequenceOfUint8
}
```

In this type:

- `countryOnly` is a `CountryOnly` as defined above.
- `region` identifies one or more regions within the country. If `countryOnly` indicates the United States of America, the values in this field identify the state or statistically equivalent entity using the integer version of the 2010 FIPS codes as provided by the U.S. Census Bureau (see normative references in Clause 2). For other values of `countryOnly`, the meaning of `region` is not defined in this version of this standard.

#### 6.4.25 CountryAndSubregions

```
CountryAndSubregions ::= SEQUENCE {
    country           CountryOnly,
    regionAndSubregions SequenceOfRegionAndSubregions
}
```

In this type:

- `country` is a `CountryOnly` as defined above.
- `regionAndSubregions` identifies one or more subregions within `country`. If `country` indicates the United States of America, the values in this field identify the county or county equivalent entity using the integer version of the 2010 FIPS codes as provided by the U.S. Census

Bureau (see normative references in Clause 2). For other values of `country`, the meaning of `regionAndSubregions` is not defined in this version of this standard.

#### Critical information fields:

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not support the number of `RegionAndSubregions` in the `CountryAndSubregions` when verifying a signed SPDU shall indicate that the signed SPDU is invalid. A compliant implementation shall support `CountryAndSubregions` containing at least eight `RegionAndSubregions` entries.

#### 6.4.26 RegionAndSubregions

```
RegionAndSubregions ::= SEQUENCE {
    region          Uint8,
    subregions      SequenceOfUint16
}

SequenceOfRegionAndSubregions ::= SEQUENCE OF RegionAndSubregions
```

In this type:

- `region` identifies a region within a country as specified under `CountryAndRegions`.
- `subregions` identifies one or more subregions as specified under `CountryAndSubregions`.

#### 6.4.27 SubjectAssurance

```
SubjectAssurance ::= OCTET STRING (SIZE(1))
```

This field contains the certificate holder's assurance level, which indicates the security of both the platform and storage of secret keys as well as the confidence in this assessment.

This field is encoded as defined in Table 1, where “A” denotes bit fields specifying an assurance level, “R” reserved bit fields, and “C” bit fields specifying the confidence.

**Table 1—Bitwise encoding of subject assurance**

Bit number	7	6	5	4	3	2	1	0
Interpretation	A	A	A	R	R	R	C	C

In Table 1, bit number 0 denotes the least significant bit. Bit 7 to bit 5 denote the device's assurance levels, bit 4 to bit 2 are reserved for future use, and bit 1 and bit 0 denote the confidence.

The specification of these assurance levels as well as the encoding of the confidence levels is outside the scope of the present document. It can be assumed that a higher assurance value indicates that the holder is more trusted than the holder of a certificate lower assurance value and the same confidence value.

NOTE—This field was originally specified in ETSI TS 103 097 [B7] and future uses of this field are anticipated to be consistent with future versions of that document.

#### **6.4.28 PsidSsp**

```

PsidSsp ::= SEQUENCE {
    psid
        Psid,
    ssp
        ServiceSpecificPermissions OPTIONAL
}

SequenceOfPsidSsp ::= SEQUENCE OF PsidSsp

```

This structure represents the permissions that the certificate holder has with respect to data for a single application area, identified by a Psid. If the ServiceSpecificPermissions field is omitted, it indicates that the certificate holder has the default permissions associated with that Psid. These permissions are Psid-specific. See Annex C for further discussion.

#### **6.4.29 ServiceSpecificPermissions**

```

ServiceSpecificPermissions ::= CHOICE {
    opaque
        OCTET STRING (SIZE(0..MAX)),
    ...
}

```

This structure represents the Service Specific Permissions (SSP) relevant to a given entry in a PsidSsp. The meaning of the SSP is specific to the associated Psid. See Annex C for further discussion.

#### **Critical information fields:**

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

#### **6.4.30 PsidGroupPermissions**

```

PsidGroupPermissions ::= SEQUENCE {
    subjectPermissions SubjectPermissions,
    minChainDepth      INTEGER DEFAULT 1, -- 0 for enrolment certs
    chainDepthRange    INTEGER DEFAULT 0, -- max depth = min + range
    eeType             EndEntityType DEFAULT {app}
}

```

```
SequenceOfPsidGroupPermissions ::= SEQUENCE OF PsidGroupPermissions
```

This structure states the permissions that a certificate holder has with respect to issuing and requesting certificates for a particular set of PSIDs. In this structure:

- `subjectPermissions` indicates PSIDs and SSP Ranges covered by this field.
- `minChainDepth` and `chainDepthRange` indicate how long the certificate chain from this certificate to the end-entity certificate is permitted to be. The length of the certificate chain is measured from the certificate issued by this certificate to the end-entity certificate in the case of `certIssuePermissions` and from the certificate requested by this certificate to the end-entity certificate in the case of `certRequestPermissions`; a length of 0 therefore indicates that the certificate issued or requested is an end-entity certificate. The length is permitted to be (a) greater than or equal to `minChainDepth` certificates and (b) less than or equal to `minChainDepth` +

chainDepthRange certificates. The value  $-1$  for chainDepthRange is a special case: if the value of chainDepthRange is  $-1$  that indicates that the certificate chain may be any length equal to or greater than minChainDepth. See the examples below for further discussion.

- eeType takes one or more of the values app and enrol and indicates the type of certificates or requests that this instance of PsidGroupPermissions in the certificate is entitled to authorize. If this field indicates app, the chain ends may end in an authorization certificate, i.e., a certificate in which these permissions appear in an appPermissions field. If this field indicates enrol, the chain ends may end in an enrolment certificate, i.e., a certificate in which these permissions appear in a certReqPermissions permissions field), or both. Different instances of PsidGroupPermissions within a ToBeSignedCertificate may have different values for eeType.

#### **Examples:**

- An enrolment certificate has an instance of this field with minChainDepth equal to  $0$ , chainDepthRange equal to  $0$ , and eeType equal to app (because the enrolment certificate is used to request authorization certificates).
- A certificate for a CA that issues authorization certificates, i.e., certificates containing an appPermissions field (see 5.1.1), might have an instance of this field for a given PSID/SSP combination with minChainDepth equal to  $1$ , chainDepthRange equal to  $0$ , and eeType equal to app. This indicates that it is entitled to issue end-entity certificates for those PSIDs but not to sign application messages.
- A certificate for an intermediate CA might have an instance of this field for a given PSID/SSP combination with minChainDepth equal to  $2$ , chainDepthRange equal to  $0$ , and eeType equal to app. This indicates that it is entitled to issue EECA certificates for those PSIDs but not to issue end-entity certificates directly.
- A certificate for a root CA might have an instance of this field for a given PSID/SSP combination with minChainDepth equal to  $3$ , chainDepthRange equal to  $-1$ , and eeType equal to (app, enrol). This indicates that it is entitled to issue ICA certificates and that these ICA certificates are entitled to appear in chains that lead to both authorization certificates and enrolment certificates.

#### **6.4.31 SubjectPermissions**

```
SubjectPermissions ::= CHOICE {
    explicit      SequenceOfPsidSspRange,
    all           NULL,
    ...
}
```

This indicates the PSIDs and associated SSPs for which certificate issuance or request permissions are granted by a PsidGroupPermissions structure. If this takes the value explicit, the enclosing PsidGroupPermissions structure grants certificate issuance or request permissions for the indicated PSIDs and SSP Ranges. If this takes the value all, the enclosing PsidGroupPermissions structure grants certificate issuance or request permissions for all PSIDs not indicated by other PsidGroupPermissions in the same certIssuePermissions or certRequestPermissions field.

#### **Critical information fields:**

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

- If present, `explicit` is a critical information field as defined in 5.2.5. An implementation that does not support the number of `PsidSspRange` in `explicit` when verifying a signed SPDU shall indicate that the signed SPDU is invalid. A compliant implementation shall support `explicit` fields containing at least eight entries.

#### **6.4.32 EndEntityType**

```
EndEntityType ::= BIT STRING {app (0), enrol (1) } (SIZE (8))
```

This type indicates which type of permissions may appear in end-entity certificates the chain of whose permissions passes through the `ItsSspDepthRange` field containing this value. If `app` is indicated, the end-entity certificate may contain an `appPermissions` field. If `enrol` is indicated, the end-entity certificate may contain a `certRequestPermissions` field.

#### **6.4.33 PsidSspRange**

```
PsidSspRange ::= SEQUENCE {
    psid                  Psid,
    sspRange      OPTIONAL
}
```

```
SequenceOfPsidSspRange ::= SEQUENCE OF PsidSspRange
```

This structure represents the certificate issuing or requesting permissions of the certificate holder with respect to one particular set of application permissions. In this structure:

- `psid` identifies the application area.
- `sspRange` identifies the SSPs associated with that PSID for which the holder may issue or request certificates. If `sspRange` is omitted, the holder may only issue or request certificates for the default SSP for that `psid`.

#### **6.4.34 SspRange**

```
SspRange ::= CHOICE {
    opaque            SequenceOfOctetString,
    all               NULL,
    ...
}
```

This structure identifies the SSPs associated with a PSID for which the holder may issue or request certificates.

- If the choice indicated is `opaque`, the certificate holder may issue or request certificates with the listed SSPs for that PSID.
- If the choice indicated is `all`, the holder may issue or request certificates for the any SSP for that PSID.

An SSP associated with a given PSID in a subordinate certificate is consistent with the SspRange associated with that PSID in the issuing certificate if one of the following hold:

- The issuing certificate SspRange is of type opaque and one of the entries in the range exactly matches the SSP in the subordinate certificate.
- The issuing certificate SspRange is of type all.

An SspRange associated with a given PSID in a subordinate certificate is consistent with the SspRange associated with that PSID in an issuing certificate if one of the following hold:

- The issuing certificate SspRange is of type opaque and all of the entries in the subordinate certificate's SspRange exactly match an entry in the issuing certificate's SspRange.
- The issuing certificate SspRange is of type all.

#### Critical information fields:

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE when verifying a signed SPDU shall indicate that the signed SPDU is invalid.
- If present, opaque is a critical information field as defined in 5.2.5. An implementation that does not support the number of OCTET STRINGS in opaque when verifying a signed SPDU shall indicate that the signed SPDU is invalid. A compliant implementation shall support opaque fields containing at least eight entries.

#### 6.4.35 VerificationKeyIndicator

```
VerificationKeyIndicator ::= CHOICE {
    verificationKey          PublicVerificationKey,
    reconstructionValue      EccP256CurvePoint,
    ...
}
```

The contents of this field depend on whether the certificate is an implicit or an explicit certificate.

- verificationKey is included in explicit certificates. It contains the public key to be used to verify signatures generated by the holder of the Certificate.
- reconstructionValue is included in implicit certificates. It contains the reconstruction value, which is used to recover the public key as specified in SEC 4 and 5.3.2.

#### 6.4.36 PublicVerificationKey

```
PublicVerificationKey ::= CHOICE {
    ecdsaNistP256           EccP256CurvePoint,
    ecdsaBrainpoolP256r1     EccP256CurvePoint,
    ...
}
```

This structure represents a public key and states with what algorithm the public key is to be used. Cryptographic mechanisms are defined in 5.3.

An EccP256CurvePoint within a PublicVerificationKey structure is invalid if it indicates the choice x-only.

#### **Critical information fields:**

- If present, this is a critical information field as defined in 5.2.5. An implementation that does not recognize the indicated CHOICE when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

## **7. Certificate revocation lists (CRLs) and the CRL Verification Entity**

### **7.1 General**

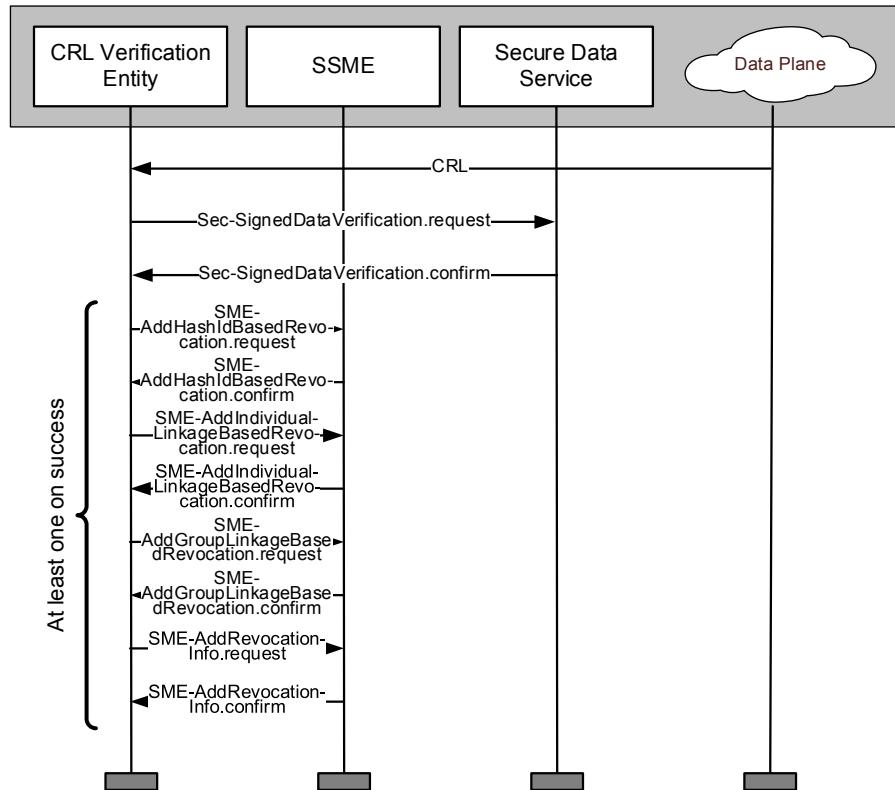
Clause 7 specifies the certificate revocation list (CRL) Verification Entity. This is a service that, by invoking the SDS and SSME, updates local stores of the revocation information whose use is specified in 5.1.3. In this clause:

- 7.2 specifies the CRL Verification Entity.
- 7.3 specifies the format of an IEEE 1609.2 CRL.
- 7.4 specifies the use of IEEE 1609.2 mechanisms to authenticate an IEEE 1609.2 CRL, using the IEEE 1609.2 security profile framework specified in C.2.

### **7.2 CRL Verification Entity specification**

The CRL Verification Entity processes and stores incoming CRLs. It receives the CRLs by mechanisms outside the scope of this standard. It shall verify a received CRL as valid by the criteria of 7.4. If the CRL is valid, the CRL Verification Entity shall pass the revocation information contained in the CRL to the SDEE for storage. Storage of revocation information in this standard is supported by the primitives SSME-AddHashIdBasedRevocation.request, SSME-AddIndividualLinkageBasedRevocation.request, SSME-AddGroupLinkageBasedRevocation.request, and SSME-AddRevocationInfo.request.

CRL Verification Entity activities are illustrated in Figure 13.



**Figure 13—CRL Verification Entity**

## 7.3 Data structures

### 7.3.1 General

Subclause 7.3 specifies the CRL contents using ASN.1. Subclause B.2 provides the complete ASN.1 module for CRLs. In the case of a conflict, 7.3 takes precedence.

For linkage ID-based CRLs, the CRL encodes the information fields specified in 5.1.3.4. Rather than listing the information fields individually for each entry, the fields are nested to provide a more compact encoding with those fields that are anticipated to have the fewest distinct values provided on the outer layers of the nesting. In particular, the CRL design anticipates that CAs organize certificate issuance such that all certificates that are potentially on the same CRL (i.e., with the same `crlSeries` and `cracaid` values) use the same `iCert` value at the same time.

### 7.3.2 CrlContents

```

CrlContents ::= SEQUENCE {
    version          Uint8 (1),
    crlSeries        CrlSeries,
    cracaid          HashedId8,
    issueDate        Time32,
    nextCrl         Time32,
    priorityInfo     CrlPriorityInfo,
}
    
```

```

typeSpecific      CHOICE {
    fullHashCrl   ToBeSignedHashIdCrl,
    deltaHashCrl   ToBeSignedHashIdCrl,
    fullLinkedCrl  ToBeSignedLinkageValueCrl,
    deltaLinkedCrl ToBeSignedLinkageValueCrl,
    ...
}
}

```

The fields in this structure have the following meaning:

- `version` is the version number of the CRL. For this version of this standard it is 1.
- `crlSeries` represents the CRL series to which this CRL belongs. This is used to determine whether the revocation information in a CRL is relevant to a particular certificate as specified in 5.1.3.2.
- `cracaId` contains the low-order eight octets of the hash of the certificate of the Certificate Revocation Authorization CA (CRACA) that ultimately authorized the issuance of this CRL. This is used to determine whether the revocation information in a CRL is relevant to a particular certificate as specified in 5.1.3.2. In a valid signed CRL as specified in 7.4 the `cracaId` is consistent with the `associatedCraca` field in the Service Specific Permissions as defined in 7.4.3.3.
- `issueDate` specifies the time when the CRL was issued.
- `nextCrl` contains the time when the next CRL with the same `crlSeries` and `cracaId` is expected to be issued. The CRL is invalid unless `nextCrl` is strictly after `issueDate`. This field is used to set the expected update time for revocation information associated with the (`cracaId`, `crlSeries`) pair as specified in 5.1.3.6.
- `priorityInfo` contains information that assists devices with limited storage space in determining which revocation information to retain and which to discard.
- `typeSpecific` contains the CRL body:
  - `fullHashCrl` contains a full hash-based CRL, i.e., a listing of the hashes of all certificates that:
    - contain the indicated `cracaId` and `crlSeries` values, and
    - are revoked by hash, and
    - have been revoked, and
    - have not expired.
  - `deltaHashCrl` contains a delta hash-based CRL, i.e., a listing of the hashes of all certificates that:
    - contain the specified `cracaId` and `crlSeries` values, and
    - are revoked by hash, and
    - have been revoked since the previous CRL that contained the indicated `cracaId` and `crlSeries` values.
  - `fullLinkedCrl` contains a full linkage ID-based CRL, i.e., a listing of the individual and/or group linkage data for all certificates that:
    - contain the indicated `cracaId` and `crlSeries` values, and

- are revoked by linkage data, and
- have been revoked, and
- have not expired.
- deltaLinkedCrl contains a delta linkage ID-based CRL, i.e., a listing of the individual and/or group linkage data for all certificates that:
  - contain the specified cracaId and crlSeries values, and
  - are revoked by linkage data, and
  - have been revoked since the previous CRL that contained the indicated cracaId and crlSeries values.

### 7.3.3 CrlPriorityInfo

```
CrlPriorityInfo ::= SEQUENCE {
    priority          Uint8 OPTIONAL,
    ...
}
```

This data structure contains information that assists devices with limited storage space in determining which revocation information to retain and which to discard.

- priority indicates the priority of the revocation information relative to other CRLs issued for certificates with the same cracaId and crlSeries values. A higher value for this field indicates higher importance of this revocation information.

### 7.3.4 ToBeSignedHashIdCrl

```
ToBeSignedHashIdCrl ::= SEQUENCE {
    crlSerial        Uint32,
    entries         SequenceOfHashBasedRevocationInfo,
    ...
}
```

This data structure represents information about a revoked certificate.

- crlSerial is a counter that increments by 1 every time a new full or delta CRL is issued for the indicated cracaId and crlSeries values.
- entries contains the individual revocation information items.

### 7.3.5 HashBasedRevocationInfo

```
HashBasedRevocationInfo ::= SEQUENCE {
    id              HashedId10,
    expiry         Time32,
    ...
}
```

```
SequenceOfHashBasedRevocationInfo ::= SEQUENCE OF HashBasedRevocationInfo
```

In this structure:

- `id` is the CertId10 identifying the revoked certificate.
- `expiry` is the value from the `expiry` field in that certificate.

### **7.3.6 ToBeSignedLinkageValueCrl**

```
ToBeSignedLinkageValueCrl ::= SEQUENCE {
    iRev                  IValue,
    intervalWithinI       Uint8,
    individual            SequenceOfJMaxGroup OPTIONAL,
    groups                SequenceOfGroupCrlEntry OPTIONAL,
    ...
}
(WITH COMPONENTS {..., individual PRESENT} |
 WITH COMPONENTS {..., groups PRESENT})
```

In this structure:

- `iRev` is the value `iRev` used in the algorithm given in 5.1.3.4. This value applies to all linkage-based revocation information included within either `individual` or `groups`.
- `intervalWithinI` is a counter that is set to 0 for the first CRL issued for the indicated combination of `cracaid`, `crlSeries`, and `iRev`, and increments by 1 every time a new full or delta CRL is issued for the indicated `cracaid` and `crlSeries` values without changing `iRev`.
- `individual` contains individual linkage data.
- `groups` contains group linkage data.

### **7.3.7 JMaxGroup**

```
JMaxGroup ::= SEQUENCE {
    jmax                 Uint8,
    contents              SequenceOfLAGroup,
    ...
}
```

`SequenceOfJMaxGroup` ::= SEQUENCE OF `JMaxGroup`

In this structure:

- `jMax` is the value `jMax` used in the algorithm given in 5.1.3.4. This value applies to all linkage-based revocation information included within `contents`.
- `contents` contains individual linkage data.

### **7.3.8 LAGroup**

```
LAGroup ::= SEQUENCE {
    la1Id                LaId,
    la2Id                LaId,
```

```

contents      SequenceOfIMaxGroup
...
}

SequenceOfLAGroup ::= SEQUENCE OF LAGroup

```

In this structure:

- la1Id is the value *LinkageAuthorityIdentifier1* used in the algorithm given in 5.1.3.4. This value applies to all linkage-based revocation information included within contents.
- la2Id is the value *LinkageAuthorityIdentifier2* used in the algorithm given in 5.1.3.4. This value applies to all linkage-based revocation information included within contents.
- contents contains individual linkage data.

### **7.3.9 IMaxGroup**

```

IMaxGroup ::= SEQUENCE {
    iMax          Uint16,
    contents      SequenceOfIndividualRevocation
    ...
}

```

```
SequenceOfIMaxGroup ::= SEQUENCE OF IMaxGroup
```

In this structure:

- iMax indicates that for the entries in contents, revocation information need no longer be calculated once  $iCert > iMax$  as the holder is known to have no more valid certs at that point. iMax is not directly used in the calculation of the linkage values but is used to determine when revocation information can safely be deleted.
- contents contains individual linkage data.

### **7.3.10 IndividualRevocation**

```

IndividualRevocation ::= SEQUENCE {
    linkageSeed1   LinkageSeed,
    linkageSeed2   LinkageSeed
    ...
}

```

```
SequenceOfIndividualRevocation ::= SEQUENCE OF IndividualRevocation
```

In this structure:

- linkageSeed1 is the value *LinkageSeed1* used in the algorithm given in 5.1.3.4.
- linkageSeed2 is the value *LinkageSeed2* used in the algorithm given in 5.1.3.4.

### 7.3.11 GroupCrlEntry

```
GroupCrlEntry ::= SEQUENCE {
    iMax                  Uint16,
    la1Id                 LaId,
    linkageSeed1          LinkageSeed,
    la2Id                 LaId,
    linkageSeed2          LinkageSeed
    ...
}

SequenceOfGroupCrlEntry ::= SEQUENCE OF GroupCrlEntry
```

In this structure:

- *iMax* indicates that for these certificates, revocation information need no longer be calculated once  $iCert > iMax$  as the holders are known to have no more valid certs for that (*cracaId*, *crlSeries*) at that point.
- *la1Id* is the value *LinkageAuthorityIdentifier1* used in the algorithm given in 5.1.3.4. This value applies to all linkage-based revocation information included within *contents*.
- *linkageSeed1* is the value *LinkageSeed1* used in the algorithm given in 5.1.3.4.
- *la2Id* is the value *LinkageAuthorityIdentifier2* used in the algorithm given in 5.1.3.4. This value applies to all linkage-based revocation information included within *contents*.
- *linkageSeed2* is the value *LinkageSeed2* used in the algorithm given in 5.1.3.4.

### 7.3.12 LaId

```
LaId ::= OCTET STRING (SIZE(2))
```

This structure contains a LA Identifier for use in the algorithms specified in 5.1.3.4.

### 7.3.13 LinkageSeed

```
LinkageSeed ::= OCTET STRING (SIZE(16))
```

This structure contains a linkage seed value for use in the algorithms specified in 5.1.3.4.

## 7.4 CRL: 1609.2 Security envelope

### 7.4.1 General

A signed CRL is a valid Ieee1609Dot2Data whose *content* field is of type *signedData*. The Signed-DataPayload structure within the signed CRL has no *extDataHash* field and the *data* field contains an Ieee1609Dot2Data whose *content* field is of type *unsecuredData* and contains a *CrlContents*.

A signed CRL may be created via the Sec-SignedData.request primitive, passing the COER-encoded *CrlContents* as the *UnsecuredData* parameter. The fields within the *SignedData* structure and the

parameters passed to Sec-SignedData.request are specified in this subclause using the IEEE 1609.2 security profile defined in C.2.

#### **7.4.2 Consistency criteria**

A valid signed CRL meets the validity criteria of Clause 5. In addition, as discussed in 5.1.3.2 and illustrated in Figure 9, a valid signed CRL also meets one of the following conditions:

- The CRL was signed by the CRACA indicated by the cracaId, or
- The CRL was signed by a certificate which was issued by the CRACA indicated by the cracaId

#### **7.4.3 Service Specific Permissions and associated consistency criteria**

##### **7.4.3.1 General**

The following Service Specific Permissions structure is defined for use by a CRL signer. These subclauses provide a specification of each data type in ASN.1.

CRL signing is identified by the PSID value allocated for CRL signing in IEEE Std 1609.12. The SSP shall be encoded with COER when included in the ServiceSpecificPermissions field of a certificate.

##### **7.4.3.2 CrlSsp**

```
CrlSsp ::= SEQUENCE {  
    version          Uint8(1),  
    associatedCraca CracaType,  
    crls            PermissibleCrls,  
    ...  
}
```

In this type:

- *version* is the version number of the SSP and is 1 for this version of the SSP.
- *associatedCraca* identifies the relationship between this certificate and the CRACA. If *associatedCraca* = *isCraca*, this certificate is the CRACA certificate and signs CRLs for certificates which chain back to this certificate. If *associatedCraca* = *issuerIsCraca*, the issuer of this certificate is the CRACA and this certificate may sign CRLs for certificates which chain back to its issuer.
- *crls* identifies what type of CRLs may be issued by the certificate holder.

##### **7.4.3.3 CracaType**

```
CracaType ::= ENUMERATED {isCraca, issuerIsCraca}
```

This type is used to determine the validity of the `cracaId` field in the `CrlContents` structure.

- If this takes the value `isCraca`, the `cracaId` field in the `CrlContents` structure is invalid unless it indicates the certificate that signs the CRL.
- If this takes the value `issuer`, the `isCracaDelegate` field in the `CrlContents` structure is invalid unless it indicates the certificate that issued the certificate that signs the CRL.

#### 7.4.3.4 PermissibleCrls

```
PermissibleCrls ::= SEQUENCE OF CrlSeries
```

This type is used to determine the validity of the `crlSeries` field in the `CrlContents` structure. The `crlSeries` field in the `CrlContents` structure is invalid unless that value appears as an entry in the `SEQUENCE` contained in this field.

#### 7.4.4 Security profile

##### 7.4.4.1 IEEE 1609.2 security profile identification

Field	Value	Notes
<code>Name</code>	“IEEE 1609.2 security profile for Certificate Revocation List”	
<code>PSIDs</code>	The value indicated in IEEE Std 1609.12 for “Certificate Revocation List Application”	
<code>Other considerations</code>		

##### 7.4.4.2 Sending

This is for information only; this standard does not specify a CRL generation application.

Field	Value	Notes
<code>Sign Data</code>	True	All CRLs are signed
<code>Signed Data in Payload</code>	True	CRL is contained within the encapsulating <code>SignedData</code>
<code>External Data</code>	False	
<code>External Data Source</code>	N/a	
<code>External Data Hash Algorithm</code>	N/a	
<code>Set Generation Time in Security Headers</code>	False	Included in CRL payload
<code>Set Generation Location in Security Headers</code>	False	Not used
<code>Set Expiry Time in Security Headers</code>	False	Included in CRL payload
<code>Signed SPDU Lifetime</code>	N/a	Not used as expiry is not used
<code>Signer Identifier Policy Type</code>	Simple	
<code>Simple Signer Identifier Policy: Minimum InterCert Time</code>	All	
<code>Simple Signer Identifier Policy: Exceptions</code>	None	
<code>Simple Signer Identifier Policy: Signer Identifier Cert Chain Length</code>	-1	
<code>Text Signer Identifier Policy</code>	N/a	
<code>Sign With Fast Verification</code>	True	Allow fast verification
<code>EC Point Format</code>	Compressed	Reduce key size
<code>p2pcd_useInteractiveForm</code>	False	Full cert chain is attached
<code>p2pcd_maxResponseBackoff</code>	N/a	
<code>p2pcd_responseActiveTimeout</code>	N/a	

<i>p2pcd requestActiveTimeout</i>	N/a	
<i>p2pcd observedRequestTimeout</i>	N/a	
<i>p2pcd currentlyUsedTriggerCertificateTime</i>	N/a	
<i>p2pcd responseCountThreshold</i>	N/a	
<i>Repeat Signed SPDUs</i>	N/a	CRL transmission is different from the CRL signing application, so this field doesn't apply
<i>Time Between Signing</i>	N/a	CRL transmission is different from the CRL signing application, so this field doesn't apply
<i>Encrypt Data</i>	False	

#### 7.4.4.3 Receiving

Field	Value	Notes
<i>Use Preprocessing</i>	False	CRLs have the full certificate chain and do not use P2PCD, so no need for preprocessing
<i>Verify Data</i>	True	
<i>Maximum Certificate Chain Length</i>	8	
<i>Relevance: Replay</i>	False	Replayed CRLs are not an attack
<i>Relevance: Generation Time in Past</i>	False	Generation time of CRL is not used to decide whether or not it is relevant
<i>Validity Period</i>	N/a	
<i>Relevance: Generation Time in Future</i>	False	CRLs could conceivably be issued to take effect in the future
<i>Acceptable Future Data Period</i>	N/a	
<i>Generation Time Source</i>	Payload	Generation time is compared with CRL signing cert validity period
<i>Relevance: Expiry Time</i>	N/a	CRLs do not expire as such—even if another CRL is issued, the previous CRL is still valid for use
<i>Expiry Time Source</i>	Payload	
<i>Consistency: Generation Location</i>	False	Generation location is irrelevant for CRLs
<i>Relevance: Generation Location Distance</i>	N/a	
<i>Validity Distance</i>	N/a	
<i>Generation Location Source</i>	N/a	
<i>Overdue CRL Tolerance</i>	1 week	The revocation list for a CRL signer should never be overdue as the CRL for a CRL signer can be distributed by the same mechanism as the CRL signed by that CRL signer
<i>Relevance: Certificate Expiry</i>	True	A CRL signed with an expired certificate should not be accepted
<i>Encrypted Data</i>	False	CRLs are not encrypted

#### 7.4.4.4 Security management

Field	Value	Notes
<i>Signing Key Algorithm</i>	ecdsaNistP256, ecdsaBrainpoolP 256r1	
<i>Encryption Algorithm</i>	n/a	
<i>Implicit or Explicit Certificates</i>	Implicit	
<i>EC Point Format</i>	Compressed	
<i>Supported Geographic Regions</i>	None	CRLs are not limited by geographic region
<i>Maximum Certificate Chain Length</i>	8	
<i>Use Individual Linkage ID</i>	False	CRL signers use identified certs and are revoked by hash if necessary
<i>Use Group Linkage ID</i>	False	CRL signers use identified certs and are revoked by hash if necessary
<i>Signature Algorithms in Chain or CRL</i>	ecdsaNistP256, ecdsaBrainpoolP 256r1	May be constrained by the security profile for the relevant application

#### 7.4.4.5 Other

Field	Value	Notes
<i>Fields that may be subject to policy update</i>	Overdue CRL Tolerance, Signer Identifier Cert Chain Length	

## 7.4.5 ASN.1

The following ASN.1 expresses a secured CRL consistent with the security profile above.

```
CrlPsid ::= Psid(135)

SecuredCrl ::= Ieee1609Dot2Data (WITH COMPONENTS {....,
    content (WITH COMPONENTS {
        signedData (WITH COMPONENTS {....,
            tbsData (WITH COMPONENTS {
                payload (WITH COMPONENTS {....,
                    data (WITH COMPONENTS {....,
                        content (WITH COMPONENTS {
                            unsecuredData (CONTAINING CrlContents)
                        })
                    })
                })
            })
        })
    })
}, headerInfo (WITH COMPONENTS {....,
    psid (CrlPsid),
    generationTime ABSENT,
    expiryTime ABSENT,
    generationLocation ABSENT,
    p2pcdLearningRequest ABSENT,
    missingCrlIdentifier ABSENT,
    encryptionKey ABSENT
})
}
}
}
}
})
```

## **8. Peer-to-peer certificate distribution (P2PCD)**

## 8.1 General

Clause 8 specifies peer-to-peer certificate distribution (P2PCD), which is a functionality obtained by the cooperation of the P2PCD Entity, the SSME, the SDS, and an appropriately behaving SDEE referred to as the *trigger SDEE*.

P2PCD is initiated when a device receives a signed SPDU for which WAVE Security Services are unable to construct a certificate chain due to not recognizing the issuer of the topmost certificate provided within the signed SPDU. The received SPDU is referred to as a *trigger SPDU*.

The device that received the trigger SPDU uses P2PCD learning requests to request peer devices to provide the necessary certificates to complete the chain. A P2PCD learning request is a field which the SDS inserts into SPDUs when signing them on behalf of the SDEE that received the original SPDU. P2PCD learning responses are sent as PDUs by the P2PCD Entity to P2PCD Entities on peer devices. The design of the P2PCD service includes throttling mechanisms to reduce the risk of channel flooding by limiting the number of responses to a single request.

P2PCD is supported by functionality within the SDS and the SSME as well as by the P2PCD Entity. This clause specifies interactions between all three of these entities to support P2PCD. The primitives specified in Clause 9 support these interactions.

In this clause:

- 8.2 specifies P2PCD operations.
- 8.3 specifies the P2PCD Entity.
- 8.4 specifies data structures and encoding for PDUs created and consumed by the P2PCD Entity.

PDUs created and consumed by the P2PCD Entity are not signed or encrypted and so are not encapsulated within an IEEE1609Dot2Data.

## 8.2 P2PCD operations

### 8.2.1 General

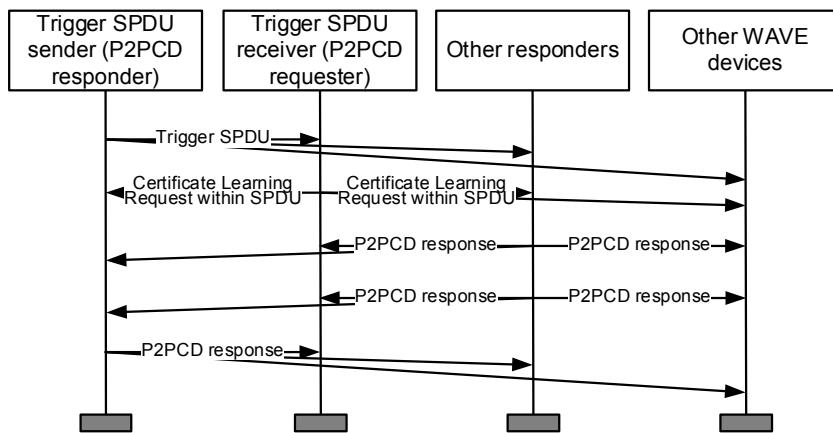
The following is an overview of P2PCD operations.

- Signed SPDUs are received by a *trigger SDEE* and processed by the SDS. In the course of this processing:
  - If the signed SPDPU indicates that the sender is using certificates issued by a CA unknown to the local SDEE, then under the conditions described in 8.2.4.1 the P2PCD request process is triggered.
  - If the signed SPDPU contains a P2PCD learning request, then under the conditions described in 8.2.4.2 the P2PCD response process is triggered.
- The P2PCD Entity (P2PCDE) monitors the data plane for incoming P2PCD learning responses. These responses are used to learn CA certificates and to determine whether or not to send responses to received requests. The P2PCDE carries out this monitoring even if its WAVE device has not recently sent a P2PCD learning request.
- In the P2PCD request process, the SDS inserts a P2PCD learning request field in signed SPDUs from the trigger SDEE. The P2PCD learning request field is defined in 6.3.9. To control SPDU size, the P2PCD learning request is only inserted under the conditions specified in 8.2.4.1.
- In the P2PCD response process, the P2PCDE is requested by the SDEE to send P2PCD learning responses. The P2PCD learning response is defined in 8.4.1 and contains the requested certificates. It is sent to a broadcast address to allow the certificates to be learned by other devices and to allow other responders to determine how many responses have been sent. To reduce the risk of the channel being flooded by responses to a single request, the P2PCD learning response is only sent under the conditions specified in 8.2.4.2, i.e., only if some threshold number of responses has not been observed since the relevant request.

An example of information flows to support P2PCD is given in the illustrative Figure 14. In the figure, each box is a WAVE device or set of WAVE devices, with each device hosting the functional entities specified above. A breakdown of the information flows showing the roles played by each functional entity is given in the illustrative Figure 16.

- a) The trigger SPDU sender, a WAVE device, sends a trigger SPDU which is received by the other WAVE devices including:
  - 1) The trigger SPDU receiver
  - 2) Other WAVE devices that will later play a responder role
  - 3) Other WAVE devices that will later not play a responder role
- b) One of the receivers of the trigger SPDU takes on the role of P2PCD requester and sends a P2PCD learning request. This is received by all the WAVE devices.
- c) The original sender, and the other responders, all select a random backoff time and send responses once that backoff time has expired. After the third response, the threshold number is reached and no more responses are sent.

Other patterns are possible, depending on how many possible requesters hear the original trigger SPDU, how many possible responders hear the request, and the order in which the responders respond.



**Figure 14—Overview of information flows for P2PCD**

### 8.2.2 Functional entities

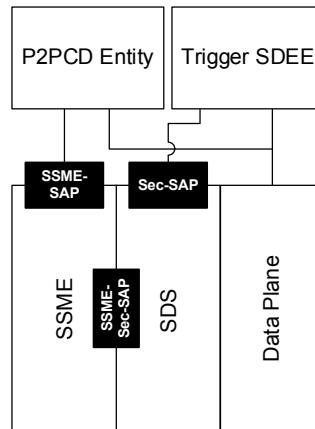
Figure 15 shows the functional entities on a device that support peer-to-peer certificate distribution.

- The data plane is used to exchange PDUs between instances of the P2PCDE and between instances of the trigger SDEE.
- The trigger SDEE sends and/or receives signed SPDUs. Fields in the signed SPDUs are used to transfer P2PCD learning requests between peer SSMEs.
- The trigger SDEE indicates support for P2PCD using parameters to primitives across the Sec-SAP. Thus, part of the specification of a SDEE is an indication of whether the SDEE acts as a trigger SDEE within P2PCD. The 1609.2 security profile (see Annex C) provides a means for an application specification to specify whether it makes use of P2PCD and if so what values are taken by the parameters defined in 8.2.3.

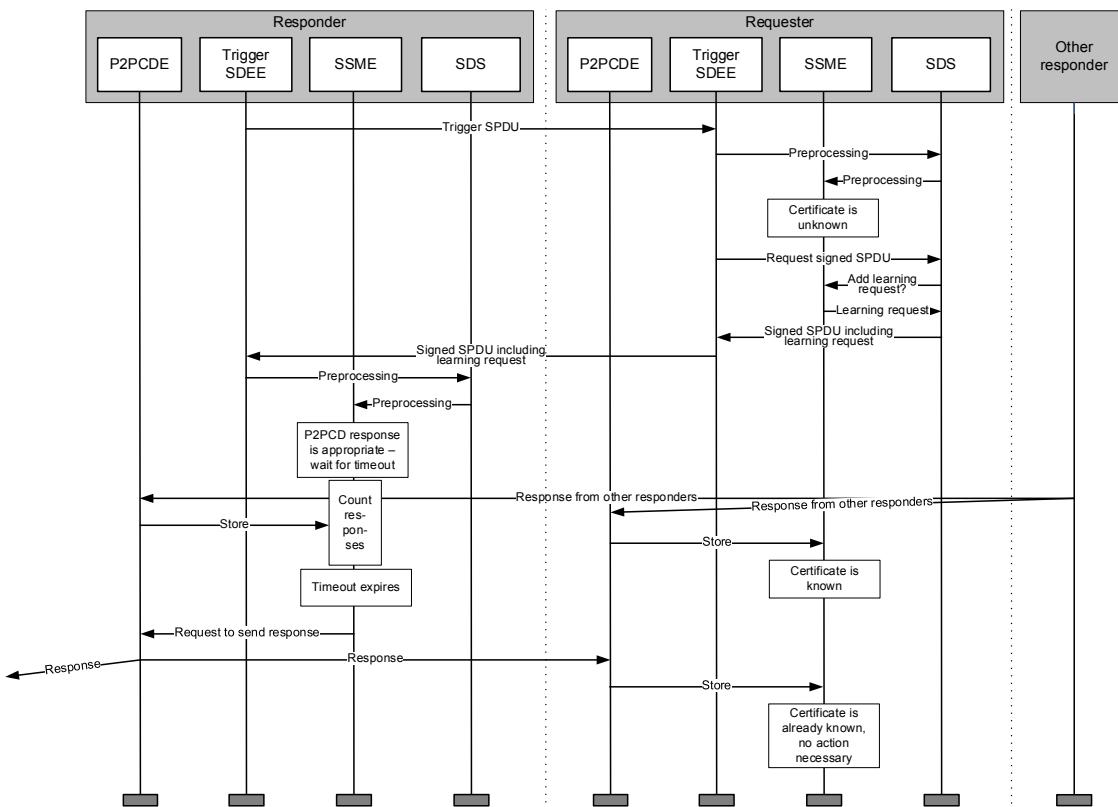
- The SDS provides the following functionality:
  - A trigger SDEE passes received signed SPDUs to the SDS via the Sec-SAP for processing to determine if P2PCD needs to be triggered, and to request that P2PCD learning requests are included in the trigger SDEE's signed SPDUs if determined to be appropriate by the SSME.
  - The SDS provides information about incoming SPDUs to the SDS via the SSME-Sec-SAP to enable it to determine whether to include P2PCD learning requests in SPDUs.
  - The SSME requests the SDS via the SSME-Sec-SAP to include P2PCD learning requests in SPDUs.
- The SSME provides the following functionality:
  - The SDS provides information about incoming SPDUs to the SDS via the SSME-Sec-SAP to enable it to determine whether to include P2PCD learning requests in SPDUs.
  - The SSME requests the SDS via the SSME-Sec-SAP to include P2PCD learning requests in SPDUs.
  - The SSME and the P2PCDE communicate via the SSME-SAP to store of certificates received via PCPCD learning response PDUs; to register the P2PCDE to send P2PCD learning responses on behalf of a particular trigger SDEE; and to request the P2PCD Entity to send a P2PCD learning response on behalf of a trigger SDEE for which it has registered.
- The P2PCD Entity registers with the SSME to receive requests to send P2PCD learning responses, sends and receives P2PCD learning responses over the data plane; and requests the SSME to store the contents of received learning responses.

A P2PCD learning request is triggered by a trigger SPDU received by the trigger SDEE, and is included in a signed SPDU generated by the same trigger SDEE. The SDS determines that incoming and outgoing SPDUs are associated with the same SDEE using the mechanisms of 4.2.2.1.

The illustrative Figure 16 shows information flows for an instance of P2PCD, showing the information flows within instances of WAVE Security Services as well as between WAVE devices.



**Figure 15—Functional entities involved in peer-to-peer certificate distribution**



**Figure 16—Overview of information flows showing functional entities**

### 8.2.3 Configuration parameters within SSME

P2PCD uses the following configuration parameters, which are managed by the SSME. These parameters may be SDEE-specific, or may be obtained from a system specification covering multiple SDEEs. They are configured by SSME-P2pcdConfiguration.request and SSME-P2pcdConfiguration.confirm. Recommended values are included in the discussion of these values in the send-side security profile in C.2.1.3.1.

- `p2pcd_useInteractiveForm` (SDEE ID  $s$ ): A Boolean indicating whether the interactive form of P2PCD is in use for the indicated SDEE. The following parameters are used only if `p2pcd_useInteractiveForm( $s$ )` is true:
  - `p2pcd_requestActiveTimeout` (SDEE ID  $s$ ): After the SSME requests the insertion of a P2PCD learning request for any particular certificate, it does not request the insertion of another P2PCD learning request for the same certificate and for the same SDEE  $s$  for at least time `p2pcd_requestActiveTimeout`.
  - `p2pcd_observedRequestTimeout` (SDEE ID  $s$ ): After the SSME observes a P2PCD learning request for any particular certificate in an incoming SPDU for  $s$ , it does not request the insertion of a P2PCD learning request for that certificate in an outgoing SPDU for  $s$  for at least time `p2pcd_requestActiveTimeout`.
  - `p2pcd_maxResponseBackoff` (SDEE ID  $s$ ): The maximum time that the SSME waits before deciding whether or not to request sending of a P2PCD learning response for a P2PCD learning request received via  $s$ .

- `p2pcd_responseActiveTimeout` (SDEE ID *s*): After the SSME triggers the response process in response to a certificate request received via *s*, it does not trigger another response process until a time equal to `p2pcd_responseActiveTimeout` has passed.
- `p2pcd_currentlyUsedTriggerCertificateTime` (SDEE ID *s*): The only requested certificates for which the SSME triggers a P2PCD learning response process are those for which, within a time indicated by `p2pcd_currentlyUsedTriggerCertificateTime`, the SDS signed a SPDU for *s* using a certificate that had the requested certificate in its chain.
- `p2pcd_responseCountThreshold` (SDEE ID *s*): The number used to determine whether `p2pcdResponseCount` is sufficiently low to allow the SSME to request generation of a P2PCD response to a particular request received via *s*.

## 8.2.4 Operations

### 8.2.4.1 Requester role

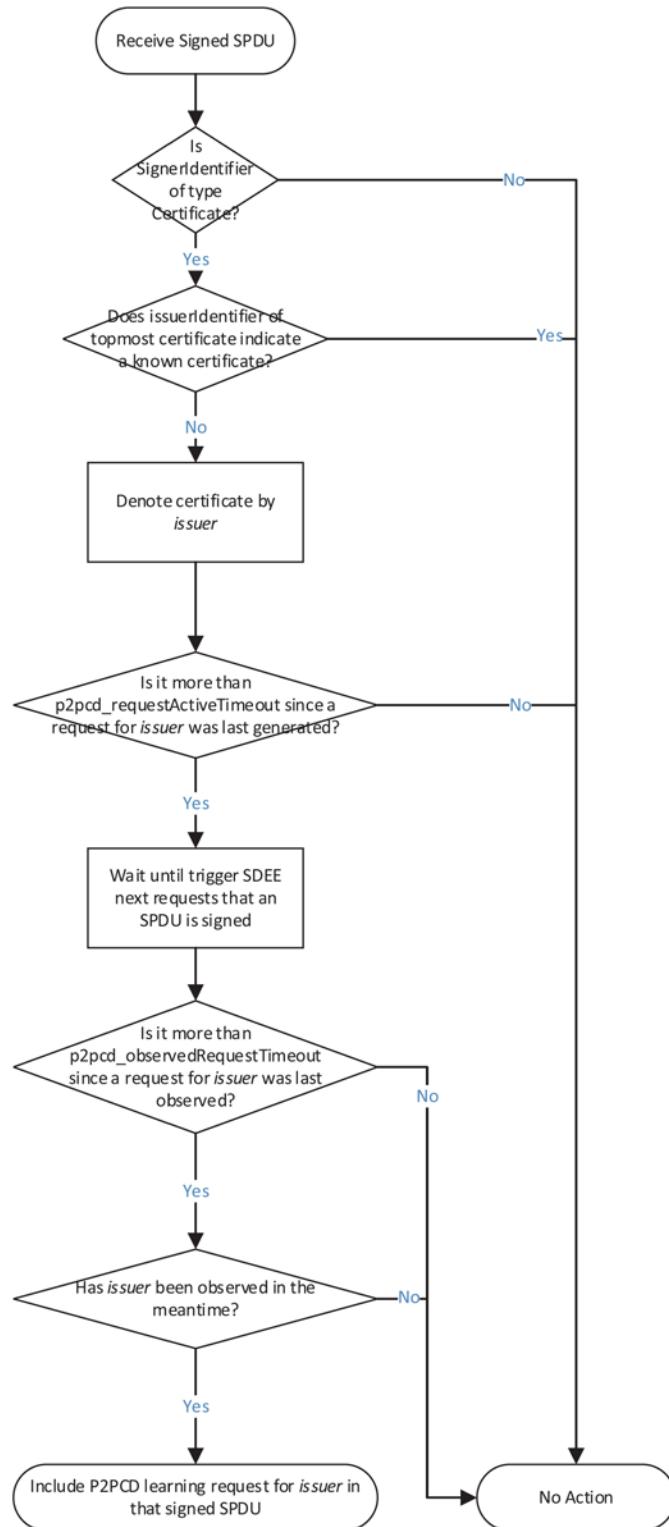
This subclause specifies requester role operations for P2PCD for a single SDEE. Subclause D.4 provides an example of how P2PCD may be implemented using the primitives defined in this standard.

- a) The P2PCD learning request process starts when a trigger SDEE requests (via `Sec-SecureData-Preprocessing.request`) that the SDS preprocesses a signed SPDU with `SignerIdentifier` of type `certificate`.
  - 1) In this case, denote by *issuer* the certificate that issued the highest certificate in the chain contained in the `SignerIdentifier`, i.e., the certificate identified by the `issuer` field in that highest certificate.
  - 2) If *issuer* indicates a certificate that is not known to the SSME, then the SSME may trigger P2PCD request processing with respect to *issuer*, unless at least one of the following exception conditions holds:
    - i) **Exception:** The SSME does not trigger request processing with respect to *issuer* if there is an active request with respect to *issuer* at the current time, i.e., it is less than a time `p2pcd_requestActiveTimeout` since the SSME last triggered request processing with respect to *issuer*.
    - ii) **Exception:** An implementation of the SSME may have a limit on the number of P2PCD learning requests that may be active simultaneously, i.e., the number of requests for which it is less than `p2pcd_requestActiveTimeout(s)` since that request was issued. If this is the case, the SSME does not trigger request processing if the number of active requests is equal to that limit. An implementation of the SSME that supports P2PCD shall support at least one active request and may support more.
- b) Upon triggering request processing with respect to *issuer*, the SSME causes the SDS to include a P2PCD learning request for *issuer* in the next SPDU *spdu* signed for the trigger SDEE, unless one of the following exceptions hold. The P2PCD learning request field is defined in 6.3.9.
  - 1) **Exception:** If the SSME has been notified of a P2PCD learning request for *issuer* in the time interval of length `p2pcd_observedRequestTimeout` before *spdu* is signed, the SSME does not cause the P2PCD learning request to be included.
  - 2) **Exception:** If there are multiple *issuer* certificates for which request processing has been triggered and for which a P2PCD learning request has not been included within `p2pcd_requestActiveTimeout`, the SSME includes a P2PCD learning request for only one of those certificates. The mechanism for selecting the certificate to be requested is

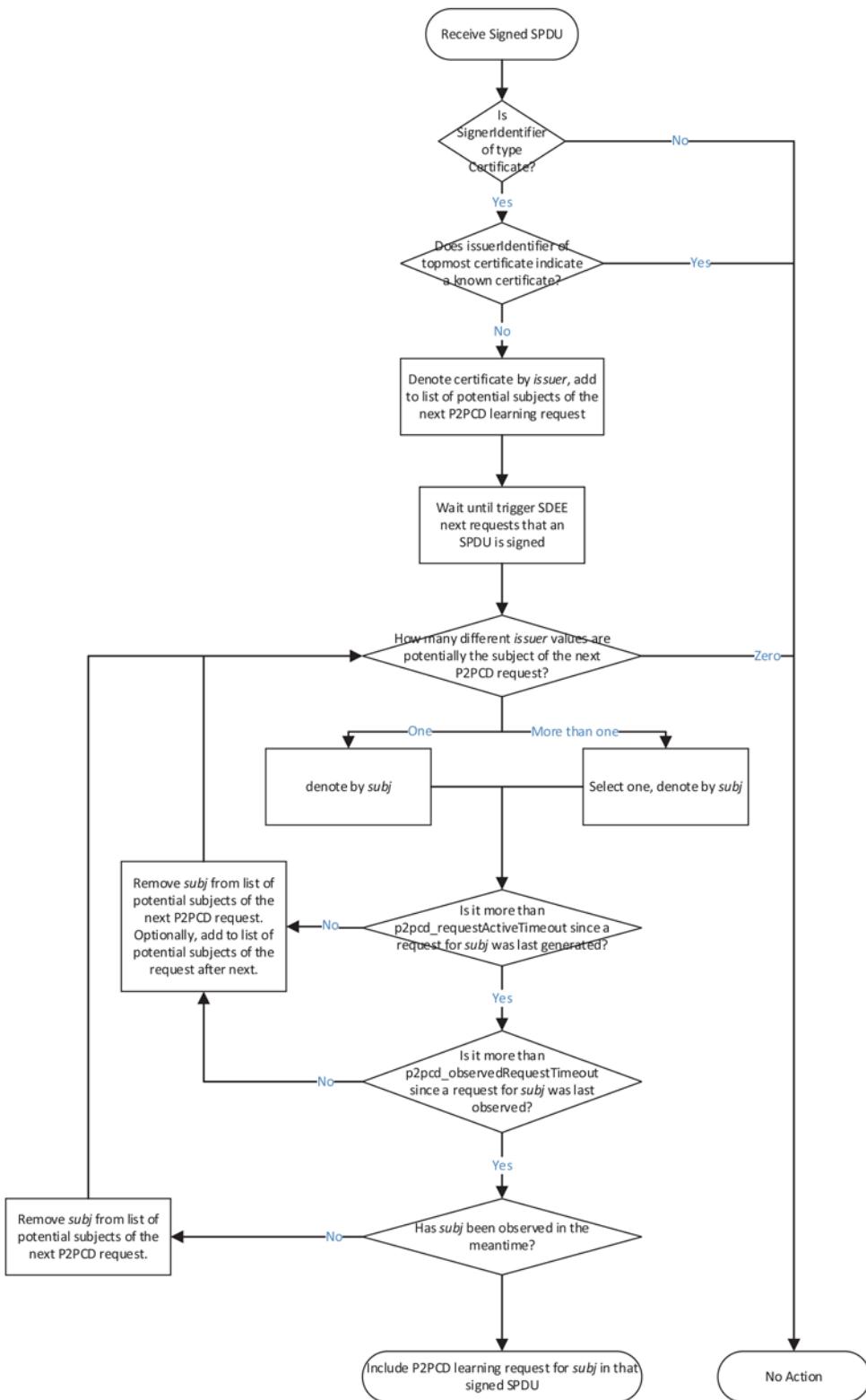
not specified in this standard. The SSME may discard the values of *issuer* for which a request was not included, or may store them for an implementation-specific period and insert a request for a stored value of *issuer* in future signed SPDUs for the trigger SDEE.

- c) When the PCPCDE receives a P2PCD learning response, it provides it to the SSME (via SSME-AddCertificate.request). The SSME stores any previously unknown certificates contained in the response, causing them to become known certificates.
- d) The SSME does not cause a P2PCD learning request to be generated for a certificate that is already known to the SSME.

Basic requester behavior, when there is only one certificate that may be the subject of the request, is illustrated in Figure 17. Requester behavior when multiple certificates may be the subject of a request is illustrated in Figure 18. Subclause D.4 provides additional figures breaking down the activities by functional elements and identifying information flows between them.



**Figure 17—P2PCD requester behavior within WAVE Security Services**



**Figure 18—P2PCD requester behavior when multiple certificates may be the subject of a request**

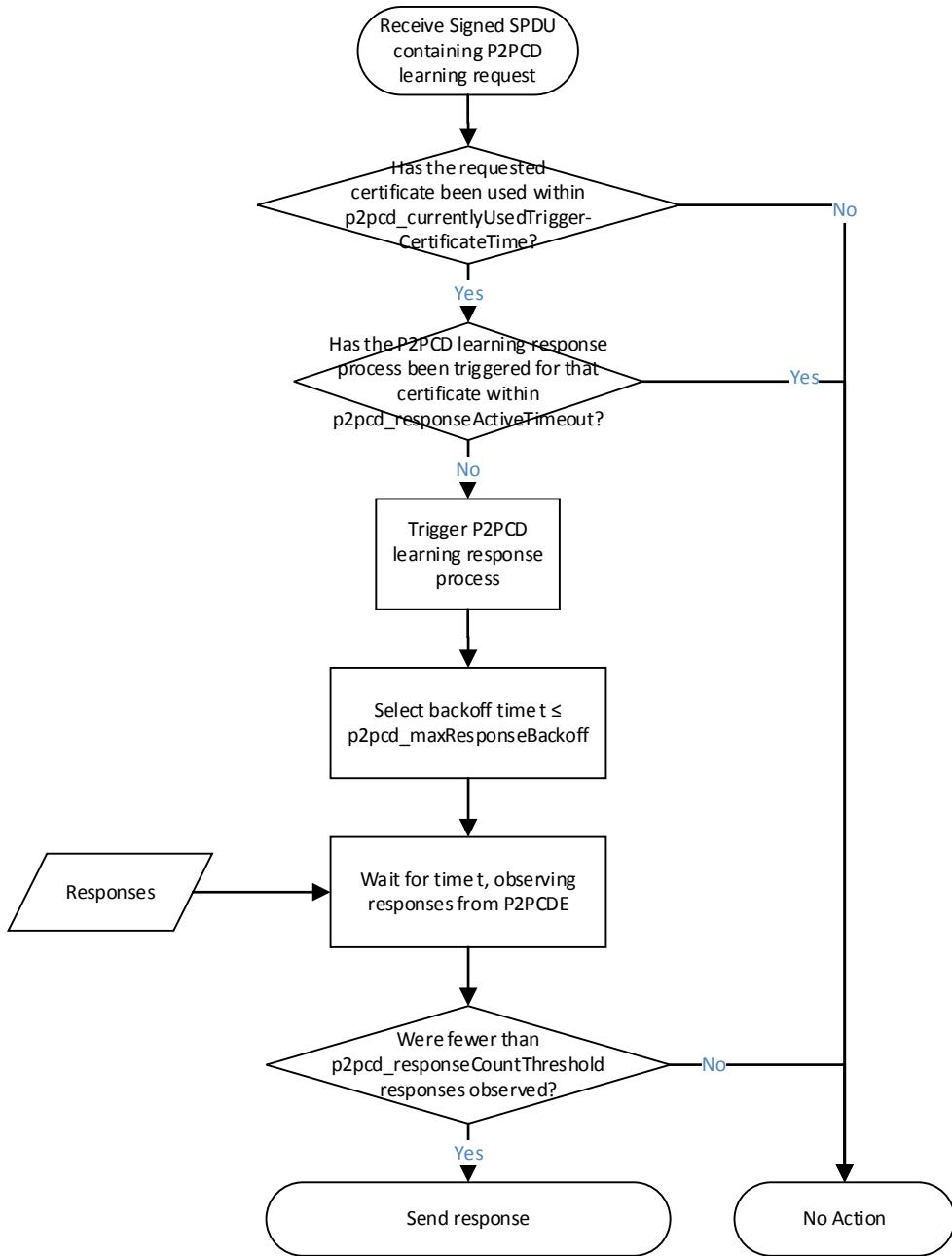
### 8.2.4.2 Responder role

This subclause specifies responder role operations for P2PCD for a single SDEE. Subclause D.4 provides an example of how P2PCD may be implemented using the primitives defined in this standard.

If the P2PCD interactive learning form is in use:

- a) The P2PCD learning response process starts when a trigger SDEE requests (via Sec-SecureData-Preprocessing.request) that the SDS preprocesses a signed SPDU containing a P2PCD learning request.
  - i) If the P2PCD learning request indicates a CA certificate that is in the chain of a certificate that has been used by the SDS to sign a SPDU within the time `p2pcd_currentlyUsedTriggerCertificateTime`, denote this by *requested*. The SSME triggers response processing with respect to *requested* unless at least one of the following exceptions hold:
    - i) **Exception:** If the current time is more than `p2pcd_responseActiveTimeout` time since the P2PCD learning response process was last triggered to respond to a request for *requested*, the SSME does not trigger response processing.
    - ii) **Exception:** If the number of active responses is more than some implementation-specific amount, the SSME does not trigger response processing. The Protocol Implementation Conformance Statement (PICS) in A.2.3.3 allows suppliers to make a statement about the numbers supported by an implementation.
- b) When the P2PCD learning response process is triggered:
  - 1) The SSME waits a random period of time less than or equal to `p2pcd_maxResponse-Backoff`. It then generates and, via SSME-P2pcdResponseGeneration.indication, requests the P2PCDE to send, a P2PCD learning response as defined in 8.4.1, unless the following exception holds.
    - i) **Exception:** If, between the triggering of the response process with respect to *requested* and the generation of the response, the P2PCDE observes a number of responses to the request greater than or equal to `p2pcd_responseCountThreshold`, then the SSME does not generate a response and does not request the P2PCDE to send any response. P2PCDE observation of responses is specified in step c) below.
- c) When the PCPCDE receives a P2PCD learning response, it provides it to the SSME (via SSME-AddCertificate.request). The SSME records certificates which are the subject of an active response process and increments the recorded number of responses to the relevant request.

Responder behavior is illustrated in Figure 19. Subclause D.4 provides additional figures breaking down the activities by functional elements and identifying information flows between them.



**Figure 19—Interactive form of P2PCD responder behavior within WAVE Security Services**

### 8.2.5 SDEE specification considerations

A complete specification of a SDEE that uses WAVE Security Services includes a specification of whether or not that SDEE uses P2PCD, and, if so, what the values are of the configuration parameters defined in 8.2.3. The IEEE 1609.2 security profile specified in Annex C may be used for this purpose. The specification need not give fixed values for the parameters; they could, for example, be obtained from a system specification covering multiple SDEEs.

## 8.2.6 Conformance

An implementation of WAVE Security Services may have a limit on the number of active P2PCD learning request or response processes that are active at any one time. An implementation of WAVE Security Services may also have a limit on the number distinct SDEEs for which it supports P2PCD. The Protocol Implementation Conformance Statement (PICS) in A.2.3.3 allows suppliers to make a statement about the numbers supported by an implementation. A conformant implementation of WAVE Security Services that supports P2PCD learning requests shall support at least one active learning request at one time and may support more. A conformant implementation of WAVE Security Services that supports P2PCD learning responses shall support at least one active learning response at one time and may support more. A conformant implementation of WAVE Security Services that supports P2PCD shall support P2PCD for at least one SDEE (i.e., shall support at least one set of the SDEE-specific configuration parameters specified in 8.2.3) and may support more.

## 8.3 P2PCD Entity specification

### 8.3.1 General

This subclause specifies the P2PCD Entity (P2PCDE). The P2PCDE:

- Registers with the SSME to send or receive responses on behalf of specific SDEEs via SSME-P2pcdResponseGenerationService.request.
- Sends P2PCD learning responses when so requested by the SSME via SSME-P2pcdResponse-Generation.indication, as specified in 8.2.4.2.
- Receives P2PCD learning responses and passes the certificates received in P2PCD learning responses to the SSME via SSME-AddCertificate.request, as specified in 8.2.4.1 and 8.2.4.2. Each certificate in each response is provided to the SSME, even if it is a duplicate of one already received, to allow the SSME to determine whether p2pcd\_responseCountThreshold has been exceeded.

An implementation of the P2PCDE shall implement receiving P2PCD learning responses and passing them to the SSME. An implementation of the P2PCDE may implement sending P2PCD learning responses.

### 8.3.2 Use within WSMP

If the P2PCD learning response is to be sent via WSMP with TPID equal to 0 or 1, the other parameters to WSM-WaveShortMessage.request shall be set as indicated. Parameters not specified in the list below, such as the channel to be used, are set as appropriate to the specific implementation. If some other TPID for WSMP is used, or if some other networking or transport protocol is used, the parameters used are determined by mechanisms out of scope for this standard.

- *Provider Service Identifier* is set to the PSID allocated for peer-to-peer distribution of security management information in IEEE Std 1609.12.
- *Peer MAC Address* is set to the broadcast MAC address.
- *WSM Data* is set to the response.

## 8.4 Data structures

### 8.4.1 P2PCD response message

#### 8.4.1.1 ASN.1 definition

The response message is defined by the following ASN.1 module:

```
IEEE1609dot2-Peer2Peer {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) management (2) peer-to-peer (1)}

--
*****
-- Data types for Peer-to-peer distribution of IEEE P1609.2 support
data
--
-- Associated with a two-byte PSID to be assigned.
-- When broadcast over WSMP, to be encoded with COER.
--
--
*****
DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
    Uint8
FROM IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) base(1) base-types(2)}

Certificate
FROM IEEE1609dot2 {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) base(1) schema(1)}
;

Ieee1609dot2Peer2PeerPDU ::= SEQUENCE {
    version          Uint8(1),
    content          CHOICE {
        caCerts       CaCertP2pPDU,
        ...
    }
}

CaCertP2pPDU ::= SEQUENCE OF Certificate

END
```

#### **8.4.1.2 Contents and encoding**

The contents of the response are as follows:

- The choice `caCerts` is indicated.
- The `caCerts` field contains an array of certificates, such that:
  - Each certificate is issued by the next certificate in the array.
  - The first certificate in the array is the one indicated by the `p2pcdLearningRequest` value `mci` to which the response message is responding (see 8.4.2).
  - The final certificate in the array was issued by a root CA.

The response is encoded with COER.

#### **8.4.2 The p2pcdLearningRequest field**

The `p2pcdLearningRequest` field is a field in the `HeaderInfo` structure of an `Ieee1609Dot2Data` of type `signed`, defined in 6.3.9.

The `p2pcdLearningRequest` value is a `HashedId3`, calculated directly from a certificate as specified in `HashedId3` or, equivalently, calculated from the `HashedId8` of a certificate by taking the low-order three bytes of the `HashedId8` value. A `p2pcdLearningRequest` corresponds to a certificate or `HashedId8` if it is the `HashedId3` derived from that certificate or `HashedId8`.

### **9. Service primitives and functions**

#### **9.1 General comments and conventions**

Clause 9 specifies mechanisms for applying 1609.2 security processing to datagrams using primitives defined at Service Access Points (SAPs). The primitives defined at each SAP are summarized in Table 1 and specified in the indicated subclause. The details of the implementation of the primitives and their exchange protocols are not otherwise specified but are left as design decisions.

Primitives are specified as (request, confirm) pairs, where the confirm primitive returns the output from WAVE Security Services obtained from processing the input provided to the corresponding request primitive, or as indications from WAVE Security Services. Any processing that produces correct output on receipt of a given set of inputs is conformant to the standard. “Correct output” is defined for each confirm primitive below.

Where parameters are identified as optional, this indicates that they may be omitted. In a .request primitive, the SDEE specification indicates via the WAVE Security Profile of Annex C which optional parameters should be omitted. In a .confirm or .indication primitive, the primitive specification indicates how WAVE Security Services determine which optional parameters to set.

**Table 1—Summary of primitives**

SAP	Primitive	Specified in
Sec	Sec-CryptomaterialHandle.request	9.3.1.1
	Sec-CryptomaterialHandle.confirm	9.3.1.2
	Sec-CryptomaterialHandle-GenerateKeyPair.request	9.3.2.1
	Sec-CryptomaterialHandle-GenerateKeyPair.confirm	9.3.2.2
	Sec-CryptomaterialHandle-StoreKeyPair.request	9.3.3.1
	Sec-CryptomaterialHandle-StoreKeyPair.confirm	9.3.3.2
	Sec-CryptomaterialHandle-StoreCertificate.request	9.3.4.1
	Sec-CryptomaterialHandle-StoreCertificate.confirm	9.3.4.2
	Sec-CryptomaterialHandle-StoreCertificateAndKey.request	9.3.5.1
	Sec-CryptomaterialHandle-StoreCertificateAndKey.confirm	9.3.5.2
	Sec-CryptomaterialHandle-Delete.request	9.3.6.1
	Sec-CryptomaterialHandle-Delete.confirm	9.3.6.2
	Sec-SymmetricCryptomaterialHandle.request	9.3.7.1
	Sec-SymmetricCryptomaterialHandle.confirm	9.3.7.2
	Sec-SymmetricCryptomaterialHandle-HashedId8.request	9.3.8.1
	Sec-SymmetricCryptomaterialHandle-HashedId8.confirm	9.3.8.2
	Sec-SymmetricCryptomaterialHandle-Delete.request	9.3.8.3
	Sec-SymmetricCryptomaterialHandle-Delete.confirm	9.3.8.4
	Sec-SignedData.request	9.3.9.1
	Sec-SignedData.confirm	9.3.9.2
	Sec-EncryptedData.request	9.3.10.1
	Sec-EncryptedData.confirm	9.3.10.2
	Sec-SecureDataPreprocessing.request	9.3.11.1
	Sec-SecureDataPreprocessing.confirm	9.3.11.2
	Sec-SignedDataVerification.request	9.3.12.1
	Sec-SignedDataVerification.confirm	9.3.12.2
	Sec-EncryptedDataDecryption.request	9.3.13.1
	Sec-EncryptedDataDecryption.confirm	9.3.13.2
SSME	SSME-CertificateInfo.request	9.4.1.1
	SSME-CertificateInfo.confirm	9.4.1.2
	SSME-AddTrustAnchor.request	9.4.2.1
	SSME-AddTrustAnchor.confirm	9.4.2.2
	SSME-AddCertificate.request	9.4.3.1
	SSME-AddCertificate.confirm	9.4.3.2
	SSME-VerifyCertificate.request	9.4.4.1
	SSME-VerifyCertificate.confirm	9.4.4.2
	SSME-DeleteCertificate.request	9.4.5.1
	SSME-DeleteCertificate.confirm	9.4.5.2
	SSME-AddHashIdBasedRevocation.request	9.4.6.1
	SSME-AddHashIdBasedRevocation.confirm	9.4.6.2
	SSME-AddIndividualLinkageBasedRevocation.request	9.4.7.1
	SSME-AddIndividualLinkageBasedRevocation.confirm	9.4.7.2
	SSME-AddGroupLinkageBasedRevocation.request	9.4.8.1
	SSME-AddGroupLinkageBasedRevocation.confirm	9.4.8.2
	SSME-AddRevocationInfo.request	9.4.9.1
	SSME-AddRevocationInfo.confirm	9.4.9.2
	SSME-RevocationInformationStatus.request	9.4.10.1
	SSME-RevocationInformationStatus.confirm	9.4.10.2
	SSME-P2pcdResponseGenerationService.request	9.4.11.1
	SSME-P2pcdResponseGenerationService.confirm	9.4.11.2
	SSME-P2pcdResponseGeneration.indication	9.4.12.1
	SSME-P2pcdConfiguration.request	9.4.13.1
	SSME-P2pcdConfiguration.confirm	9.4.13.2
SSME-Sec	SSME-Sec-ReplayDetection.request	9.5.1.1
	SSME-Sec-ReplayDetection.confirm	9.5.1.2
	SSME-Sec-IncomingP2pcdInfo.request	9.5.2.1
	SSME-Sec-IncomingP2pcdInfo.confirm	9.5.2.2
	SSME-Sec-OutgoingP2pcdInfo.request	9.5.3.1
	SSME-Sec-OutgoingP2pcdInfo.confirm	9.5.3.2

Parameters to primitives are denoted in italics and have names beginning with uppercase letters. Variables used within primitives are denoted in italics and have names beginning with lowercase letters. Fields within the data structures defined in Clause 6 are denoted in a fixed-width font.

Some primitives take parameters that are variable length strings. This standard adopts the convention that the indicated data object provides both the length and the contents of the string. The length is denoted by *String Name.length*. The contents are denoted by *String Name.contents*.

Some primitives take parameters that are keys for cryptographic operations (or that contain keys: for example, a CMH in any state other than *Initialized*). This standard adopts the convention that the encoding of the key includes an identification of the algorithm for which it is to be used. The algorithm is denoted by *Key Name.algorithm*.

Some primitives take parameters that are arrays of variables. This standard adopts the convention that when an array is passed as a parameter, the array object makes available the number of entries in the array. The length is denoted by *Array Name.length*, and individual entries are denoted by *Array Name[i]*. The first element in an array is element 0.

A primitive may take as a parameter an array of elements where each element is structured. In this case the elements of entry i are denoted by *Array Name[i].Element Name*.

Where error indications or failure responses are considered critical to the operations, they are specified within the “.confirm” primitives. Other error conditions not defined here may be indicated in such primitives.

## **9.2 Identifiers used in the interface specification**

### **9.2.1 SDEE identifier**

Each locally distinct SDEE uses a distinct SDEE identifier, an integer, to identify itself to the SDS. This standard does not specify how an implementation enforces that different SDEEs have different identifiers.

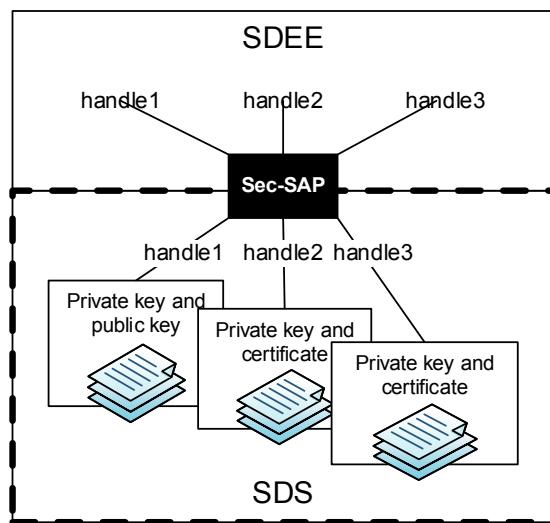
### **9.2.2 Cryptomaterial Handles**

#### **9.2.2.1 General**

For purposes of primitives defined within this standard that use public-key cryptographic operations, the model followed is that private keys and the associated public keys and certificates are stored by the SDS and referenced by the SDEE using an integer known as a Cryptomaterial Handle (CMH). A CMH in this standard is an abstract construct used to define the primitives. The interfaces defined in this standard assume that a private key and a public key, or a private key and a certificate, referenced by a CMH form a valid pair. How this is enforced in practice, how private key material is protected from being read and/or modified, how secure random numbers are obtained to support key generation, how metadata associated with cryptomaterial such as expiry time is managed, and how a CMH is deleted, is implementation specific. The CMH is an abstraction of an actual cryptographic key storage interface and not intended to provide full key management functionality. For such an interface implementers should consult an existing standard such as OASIS [B17].

An SDEE may have access to multiple CMHs at one time.

Figure 20 illustrates the CMH model.



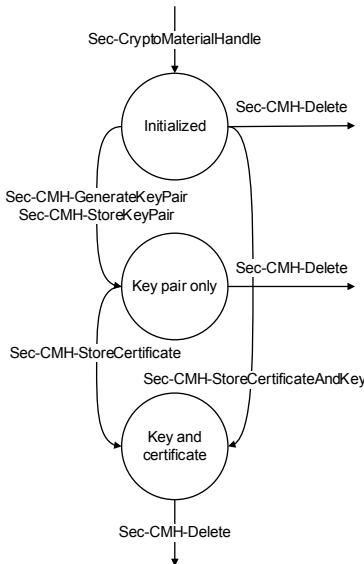
**Figure 20—SDEE and Cryptomaterial Handle SHOW PUBLIC KEY**

### 9.2.2.2 States

There are three states defined for a CMH, as follows:

- *Initialized*: A CMH in *Initialized* state does not reference any cryptomaterial.
- *Key Pair Only*: A CMH in *Key Pair Only* state references a private key and the corresponding public key.
- *Key and Certificate*: A CMH in *Key and Certificate* state references a private key and the corresponding certificate.

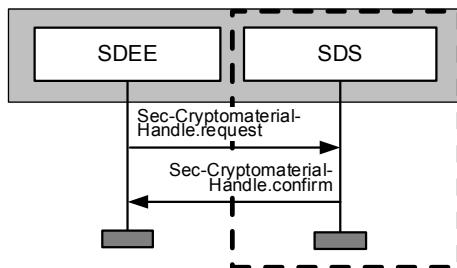
Figure 21 shows the state diagram for Cryptomaterial Handles, including an indication of the primitives that may be used to transition from one state to the next. A Cryptomaterial Handle in any state may be deleted by invoking Sec-CryptomaterialHandle-Delete.request.



**Figure 21—State diagram for Cryptomaterial Handles**

#### 9.2.2.3 Initialization

A CMH is created in the *Initialized* state as shown in Figure 22 where the dashed lines indicate functionality defined in this standard. Primitive names in the figure are abbreviated for compactness. The SDEE creates the CMH via `Sec-CryptomaterialHandle.request` and the CMH is returned via `Sec-CryptomaterialHandle.confirm`.



**Figure 22—Process flow for obtaining a new Cryptomaterial Handle**

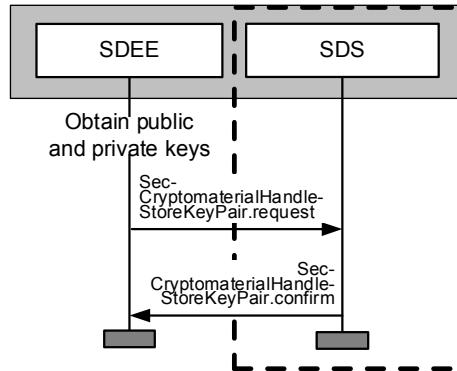
#### 9.2.2.4 Transition to *Key Pair Only* state

On invocation of `Sec-CryptomaterialHandle-StoreKeyPair.request` referencing a CMH that is in the *Initialized* state, the SDS stores the enclosed private key and place the CMH in the *Key Pair Only* state. The SDS confirms the operation back to the invoking SDEE via `Sec-CryptomaterialHandle-StoreKeyPair.confirm`. If the private and public keys referenced by the CMH do not form a valid key pair for the given cryptographic algorithm, `Sec-CryptomaterialHandle-StoreKeyPair.confirm` returns an error. Key pair validity is established for ECDSA and ECIES as specified in 5.3.7.

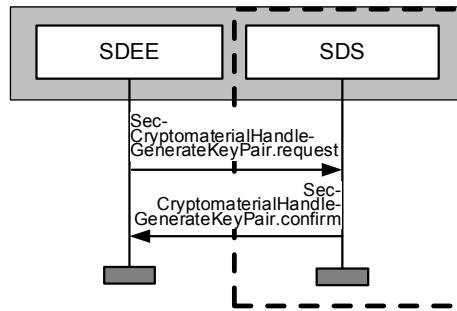
The use of `Sec-CryptomaterialHandle-StoreKeyPair.request` is illustrated in Figure 23.

On invocation of `Sec-CryptomaterialHandle-GenerateKeyPair.request` for a CMH that is in the *Initialized* state, the SDS generates a private key and public key pair, store them with the CMH and confirm the

operation back to the invoking SDEE in a Sec-CryptomaterialHandle-GenerateKeyPair.confirm, thus causing the CMH to enter the *Key Pair Only* state. This is illustrated in Figure 24.



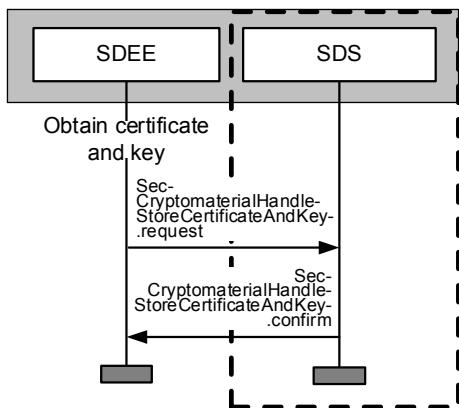
**Figure 23—Transitioning a CMH from *Initialized* to *Key Pair Only* state with externally generated keys**



**Figure 24—Transitioning a CMH from *Initialized* to *Key Pair Only* state with keys generated by the SDS**

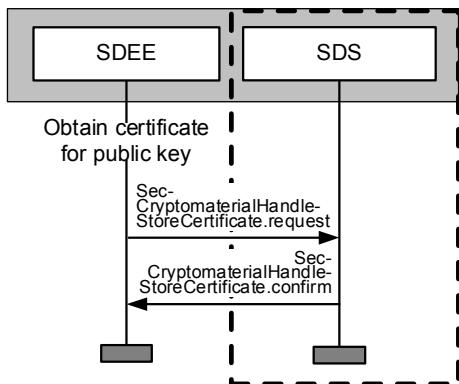
#### 9.2.2.5 Transition to *Key and Certificate* state

On invocation of Sec-CryptomaterialHandle-StoreCertificateAndKey.request for a CMH that is in the *Initialized* state, the SDS stores the enclosed private key and certificate and confirm the operation back to the invoking SDEE via Sec-CryptomaterialHandle-StoreCertificateAndKey.confirm, thus causing the CMH to enter the *Key and Certificate* state. This is illustrated in Figure 25.



**Figure 25—Transitioning a CMH directly from *Initialized* to *Key and Certificate* state**

On invocation of a Sec-CryptomaterialHandle-StoreCertificate.request for a CMH that is in the *Key Pair Only* state, the SDS stores the enclosed certificate with the CMH and confirm the operation back to the invoking SDEE in a Sec-CryptomaterialHandle-StoreCertificate.confirm, thus causing the CMH to enter the *Key and Certificate* state. This is illustrated in Figure 26.



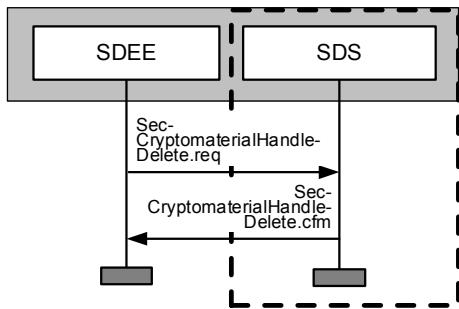
**Figure 26—Transitioning a CMH from *Initialized* to *Key and Certificate* state via *Key Pair Only* state with externally generated keys**

If the private key and the public key indicated by the certificate do not form a valid key pair for the given cryptographic algorithm, or if the certificate is not part of a valid certificate chain terminating in a trust anchor, then Sec-CryptomaterialHandle-StoreCertificate.confirm returns an error and the CMH state is undefined. Key pair validity for ECDSA and ECIES is defined in 5.3.7 (for explicit certificates) or 5.3.2 (for implicit certificates). The validity of the certificate chain is determined according to the criteria given in 5.1.2. This may also be assured using the processing specified for Sec-CryptomaterialHandle-StoreCertificate.request and Sec-CryptomaterialHandle-StoreCertificateAndKey.request.

NOTE—This standard does not provide a primitive that allows a private key to be imported to a CMH in encrypted form, but implementations may provide such an interface.

#### 9.2.2.6 Deletion

A CMH is deleted via Sec-CryptomaterialHandle-Delete.request. The deletion is confirmed via Sec-CryptomaterialHandle-Delete.confirm. This is illustrated in Figure 27.



**Figure 27—Process flow for deleting a Cryptomaterial Handle**

### 9.2.3 Symmetric Cryptomaterial Handles

#### 9.2.3.1 General

For purposes of primitives defined within this standard that use symmetric cryptographic operations, the model followed is that keys are stored by the SDS and referenced by the SDEE using an integer known as a Symmetric Cryptomaterial Handle (SCMH). A SCMH in this standard is an abstract construct used to define the primitives. How key material is protected from being read and/or modified, how secure random numbers are obtained to support key generation, how metadata associated with cryptomaterial such as expiry time is managed, and how a SCMH is deleted, is implementation specific. The SCMH is an abstraction of an actual cryptographic key storage interface and not intended to provide full key management functionality. For such an interface implementers should consult an existing standard such as OASIS [B17].

An SDEE may have access to multiple SCMHs at one time.

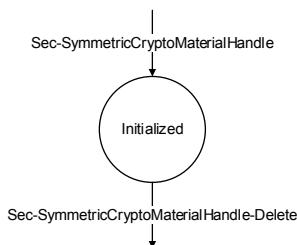
#### 9.2.3.2 State

There is a single state defined for a SCMH, as follows:

- *Initialized*: A CMH in *Initialized* state references a symmetric key.

Figure 28 shows the state diagram for Cryptomaterial Handles, including an indication of the primitives that may be used to transition from one state to the next.

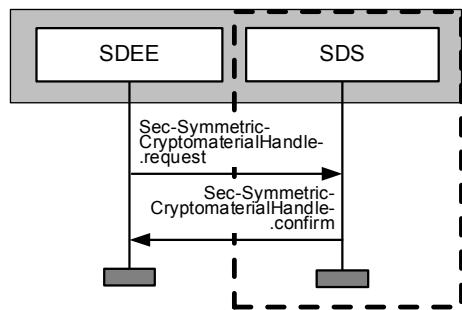
The SCMH is deleted via Sec-SymmetricCryptoMaterialHandle-Delete.request.



**Figure 28—State diagram for Symmetric Cryptomaterial Handles**

### 9.2.3.3 Initialization

A SCMH is created in the *Initialized* state as shown in Figure 29 where the dashed lines indicate functionality defined in this standard. Primitive names in the figure are abbreviated for compactness. The SDEE creates the SCMH via Sec-SymmetricCryptomaterialHandle.request and the CMH is returned via Sec-SymmetricCryptomaterialHandle.confirm. The SDEE provides the key material in the request primitive. Certain operations by the SDS can also result in the creation of a SCMH which is returned to the invoking entity; those operations are noted in Clause 9.



**Figure 29—Process flow for obtaining a new Symmetric Cryptomaterial Handle**

## 9.3 Sec SAP

### 9.3.1 Sec-CryptomaterialHandle

#### 9.3.1.1 Sec-CryptomaterialHandle.request

##### 9.3.1.1.1 Function

The primitive is used by a SDEE to request a CMH.

##### 9.3.1.1.2 Semantics of the service primitive

The primitive does not take parameters.

##### 9.3.1.1.3 When generated

The primitive is generated as needed by SDEEs.

##### 9.3.1.1.4 Effect of receipt

On receipt of this primitive the SDS generates a CMH value that it has not previously returned. The SDS returns the new CMH via the corresponding confirm primitive.

### 9.3.1.2 Sec-CryptomaterialHandle.confirm

#### 9.3.1.2.1 Function

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### 9.3.1.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle.confirm (
    Result Code
    Cryptomaterial Handle,
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success, Failure	The result of the operation
<i>Cryptomaterial Handle</i>	Integer	Any	A CMH as specified in 9.2.2

#### 9.3.1.2.3 When generated

The primitive is generated in response to Sec-CryptomaterialHandle.request.

#### 9.3.1.2.4 Effect of receipt

No behavior is specified.

### 9.3.2 Sec-CryptomaterialHandle-GenerateKeyValuePair

#### 9.3.2.1 Sec-CryptomaterialHandle-GenerateKeyValuePair.request

##### 9.3.2.1.1 Function

The primitive is used by a SDEE to request a key pair from the SDS for use with an associated CMH.

##### 9.3.2.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-GenerateKeyValuePair.request (
    Cryptomaterial Handle,
    Algorithm
)
```

Name	Type	Valid range	Description
<i>Cryptomaterial Handle</i>	Integer	Any	A CMH in <i>Initialized</i> state
<i>Algorithm</i>	Enumerated type	ecdsaBrainpoolP256r1WithSha256, ecdsaNistP256WithSha256, eciesNistp256, eciesBrainpoolP256r1	The algorithm identifier for the key pair to be generated

### 9.3.2.1.3 When generated

The primitive is generated as needed by SDEEs.

### 9.3.2.1.4 Effect of receipt

On receipt of this primitive the SDS generates a key pair for the given algorithm. The key pair is stored at the CMH provided and the CMH is transitioned to the *Key Pair Only* state. The public key is returned in the corresponding confirm primitive.

## 9.3.2.2 Sec-CryptomaterialHandle-GenerateKeyPair.confirm

### 9.3.2.2.1 Function

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

### 9.3.2.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-GenerateKeyPair.confirm (
    Result Code
    Public Key,
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success, Failure	The result of the request
<i>Public Key</i>	A public key	Any public key that is valid for the algorithm provided to the corresponding request primitive	The public key from the key pair that was generated in response to the corresponding request primitive

### 9.3.2.2.3 When generated

The primitive is generated in response to Sec-CryptomaterialHandle-GenerateKeyPair.request.

### 9.3.2.2.4 Effect of receipt

No behavior is specified.

### 9.3.3 Sec-CryptomaterialHandle-StoreKeyPair

#### 9.3.3.1 Sec-CryptomaterialHandle-StoreKeyPair.request

##### 9.3.3.1.1 Function

The primitive is used by a SDEE to request that the SDS stores a key pair generated elsewhere for use with an associated CMH.

##### 9.3.3.1.2 Semantics of the service primitive

The parameters of this primitive are as follows:

```
Sec-CryptomaterialHandle-StoreKeyPair.request (
    Cryptomaterial Handle,
    Algorithm,
    Public Key,
    Private Key
)
```

Name	Type	Valid range	Description
<i>Cryptomaterial Handle</i>	Integer	Any	A CMH as specified in 9.2.2 in <i>Initialized</i> state
<i>Algorithm</i>	Enumerated type	ecdsaBrainpoolP256r1WithSha256, ecdsaNistP256WithSha256, eciesNistp256, eciesBrainpoolP256r1	The algorithm identifier for the key pair to be stored
<i>Public Key</i>	Public key	Any public key valid for <i>Algorithm</i>	The public key to be stored
<i>Private Key</i>	Private key	Any private key valid for <i>Algorithm</i>	The private key to be stored

##### 9.3.3.1.3 When generated

The primitive is generated as needed by SDEEs.

##### 9.3.3.1.4 Effect of receipt

On receipt of this primitive the SDS verifies that the public key and private key form a valid key pair as defined in 5.3.7. If the key pair is valid, it is stored at the CMH provided and the CMH is transitioned to the *Key Pair Only* state. The public key is returned in the corresponding confirm primitive.

#### 9.3.3.2 Sec-CryptomaterialHandle-StoreKeyPair.confirm

##### 9.3.3.2.1 Function

The primitive returns the result of the corresponding request primitive.

### 9.3.3.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-StoreKeyPair.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success, Invalid key pair	The result of the operation

### 9.3.3.2.3 When generated

The primitive is generated in response to Sec-CryptomaterialHandle-StoreKeyPair.request. *Result Code* is “success” if the parameters *Public Key* and *Private Key* passed in the request primitive form a valid key pair, and “invalid key pair” if they do not.

### 9.3.3.2.4 Effect of receipt

No behavior is specified.

## 9.3.4 Sec-CryptomaterialHandle-StoreCertificate

### 9.3.4.1 Sec-CryptomaterialHandle-StoreCertificate.request

#### 9.3.4.1.1 Function

The primitive is used by a SDEE to request that the SDS stores a certificate at a specific CMH.

#### 9.3.4.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-StoreCertificate.request (
    Cryptomaterial Handle,
    Certificate,
    Private Key Transformation
)
```

Name	Type	Valid range	Description
<i>Cryptomaterial Handle</i>	Integer	Any	A CMH as specified in 9.2.2 in <i>Key Pair Only</i> state.
<i>Certificate</i>	1609.2 certificate	Any 1609.2 certificate containing a public verification key for the algorithm associated with <i>Cryptomaterial Handle</i>	The certificate to be stored.
<i>Private Key Transformation</i>	A description of a linear transformation, $y = Ax + B$	Any	A transformation to be applied to the private key to determine whether it corresponds to the public key specified by the certificate. For implicit certificates, $A$ is equal to the hash of the certificate as specified in 5.3.2 and $B$ is the private key contribution data referred to as $r$ in SEC 4, section 2.3.

#### 9.3.4.1.3 When generated

The primitive is generated as needed by SDEEs.

#### 9.3.4.1.4 Effect of receipt

On receipt of this primitive the SDS verifies that the following are true:

- a) The private key indicated by *Cryptomaterial Handle*, following the application of *Private Key Transformation*, and the public verification key indicated by *Certificate*, form a valid key pair as defined in 5.3.7 (for explicit certificates) or 5.3.2 (for implicit certificates).
- b) The certificate provided at *Certificate* is valid (for example by invoking SSME-VerifyCertificate.request).

If both statements are true, the SDS stores the updated private key and the certificate associated with the handle and the CMH is transitioned to the *Key and Certificate* state. If not, the certificate is not stored and the private key is unchanged.

#### 9.3.4.2 Sec-CryptomaterialHandle-StoreCertificate.confirm

##### 9.3.4.2.1 Function

The primitive returns the result of the corresponding request primitive.

##### 9.3.4.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-StoreCertificate.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Keys do not match Any <i>Result Code</i> value returned by SSME-VerifyCertificate.request	The result of the operation

#### 9.3.4.2.3 When generated

The primitive is generated in response to Sec-CryptomaterialHandle-StoreCertificate.request.

#### 9.3.4.2.4 Effect of receipt

No behavior is specified.

### 9.3.5 Sec-StoreCertificateAndKey

#### 9.3.5.1 Sec-CryptomaterialHandle-StoreCertificateAndKey.request

##### 9.3.5.1.1 Function

The primitive is used by a SDEE to request that the SDS stores a certificate at a specific CMH.

##### 9.3.5.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-StoreCertificateAndKey.request (
    Cryptomaterial Handle,
    Certificate,
    Private Key
)
```

Name	Type	Valid range	Description
<i>Cryptomaterial Handle</i>	Integer	Any	A CMH as specified in 9.2.2 in <i>Initialized</i> state
<i>Certificate</i>	1609.2 certificate	Any 1609.2 certificate containing a public verification key for the algorithm associated with <i>Cryptomaterial Handle</i>	The certificate to be stored
<i>Private Key</i>	A private key	Any suitable for the algorithm defined in the certificate	The private key to be stored

##### 9.3.5.1.3 When generated

The primitive is generated as needed by SDEEs.

#### **9.3.5.1.4 Effect of receipt**

On receipt of this primitive the SDS verifies that the following are true:

- a) *Private Key* and the public key indicated by *Certificate* form a valid key pair as defined in 5.3.7 (for explicit certificates) or 5.3.2 (for implicit certificates).
- b) The certificate provided at *Certificate* is valid (for example by invoking SSME-VerifyCertificate.request).

If both statements are true, the SDS stores the updated private key and the certificate associated with the handle and the CMH is transitioned to the *Key and Certificate* state. If not, the certificate is not stored and the private key is unchanged.

#### **9.3.5.2 Sec-CryptomaterialHandle-StoreCertificateAndKey.confirm**

##### **9.3.5.2.1 Function**

The primitive returns the result of the corresponding request primitive.

##### **9.3.5.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-StoreCertificate.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Keys do not match Any <i>Result Code</i> value returned by SSME-VerifyCertificate.request	The result of the operation

##### **9.3.5.2.3 When generated**

The primitive is generated in response to Sec-CryptomaterialHandle-StoreCertificate.request.

##### **9.3.5.2.4 Effect of receipt**

No behavior is specified.

### **9.3.6 Sec-CryptomaterialHandle-Delete**

#### **9.3.6.1 Sec-CryptomaterialHandle-Delete.request**

##### **9.3.6.1.1 Function**

The primitive is used by a SDEE to request deletion of a CMH and the associated cryptographic material.

##### **9.3.6.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle-Delete.request (
    Cryptomaterial Handle,
)
```

Name	Type	Valid range	Description
<i>Cryptomaterial Handle</i>	Integer	Any	A CMH as specified in 9.2.2 in any state

##### **9.3.6.1.3 When generated**

The primitive is generated as needed by SDEEs.

##### **9.3.6.1.4 Effect of receipt**

On receipt of this primitive the SDS deletes the linkage between the CMH and any cryptomaterial that it references. The SDS may also delete the cryptomaterial itself.

#### **9.3.6.2 Sec-CryptomaterialHandle-Delete.confirm**

##### **9.3.6.2.1 Function**

The primitive confirms the operation of the Sec-CryptomaterialHandle-Delete.request primitive.

##### **9.3.6.2.2 Semantics of the service primitive**

The primitive does not take parameters.

##### **9.3.6.2.3 When generated**

The primitive is generated in response to Sec-CryptomaterialHandle-Delete.request.

### 9.3.6.2.4 Effect of receipt

No behavior is specified.

## 9.3.7 Sec-SymmetricCryptomaterialHandle

### 9.3.7.1 Sec-SymmetricCryptomaterialHandle.request

#### 9.3.7.1.1 Function

The primitive is used by a SDEE to request a Symmetric Cryptomaterial Handle (SCMH) for symmetric keying material.

#### 9.3.7.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-SymmetricCryptomaterialHandle.request (
    Algorithm,
    Generate,
    Key Bytes (optional)
)
```

Name	Type	Valid range	Description
<i>Algorithm</i>	Enumerated	aes128Ccm	An identifier of the algorithm for the symmetric key as specified in 6.3.19.
<i>Generate</i>	Boolean	True, False	Indicates whether the SDS is being asked to generate the key (True) or the key material is being passed (False).
<i>Key Bytes</i>	Octet string	An octet string of the length appropriate for the algorithm indicated in <i>Algorithm</i> ; for AES-CCM, 16 bytes	The keying material to be used. Provided if <i>Generate</i> is False.

#### 9.3.7.1.3 When generated

The primitive is generated as needed by SDEEs.

#### 9.3.7.1.4 Effect of receipt

On receipt of this primitive the SDS generates a SCMH value that it has not previously returned. If *Generate* is true, the SDS generates the symmetric key material and store it at SCMH. If *Generate* is false, the SDS stores the key bytes provided via *Key Bytes* as the key material at SCMH. The SDS returns the new CMH via the corresponding confirm primitive.

### **9.3.7.2 Sec-SymmetricCryptomaterialHandle.confirm**

#### **9.3.7.2.1 Function**

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### **9.3.7.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-CryptomaterialHandle.confirm (
    Result Code,
    Symmetric Cryptomaterial Handle,
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Bad parameters Failure	The result of the operation
<i>Symmetric Cryptomaterial Handle</i>	Integer	Any	A CMH as specified in 9.2.3

#### **9.3.7.2.3 When generated**

The primitive is generated in response to Sec-CryptomaterialHandle.request.

#### **9.3.7.2.4 Effect of receipt**

No behavior is specified.

### **9.3.8 Sec-SymmetricCryptomaterialHandle-HashedId8**

#### **9.3.8.1 Sec-SymmetricCryptomaterialHandle-HashedId8.request**

##### **9.3.8.1.1 Function**

The primitive is used by a SDEE to request the HashedId8 of the key referenced by a Symmetric Cryptomaterial Handle (SCMH), for example to include it in a PreSharedKeyRecipientInfo.

### **9.3.8.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-SymmetricCryptomaterialHandle-HashedId8.request (
    Symmetric Cryptomaterial Handle
)
```

Name	Type	Valid range	Description
<i>Symmetric Cryptomaterial Handle</i>	Integer	Integer referencing a Symmetric Cryptomaterial Handle	

### **9.3.8.1.3 When generated**

The primitive is generated as needed by SDEEs.

### **9.3.8.1.4 Effect of receipt**

On receipt of this primitive the SDS generates the HashedId8 of the symmetric key referenced by *Symmetric Cryptomaterial Handle* and returns it via the corresponding return primitive.

## **9.3.8.2 Sec-SymmetricCryptomaterialHandle-HashedId8.confirm**

### **9.3.8.2.1 Function**

The primitive returns the value calculated in the processing specified for the corresponding request primitive.

### **9.3.8.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-SymmetricCryptomaterialHandle-HashedId8.confirm (
    HashedId8
)
```

Name	Type	Valid range	Description
HashedId8	HashedId8	Any	The HashedId8 of the key referenced by the <i>Symmetric Cryptomaterial Handle</i> provided to the corresponding request primitive

### **9.3.8.2.3 When generated**

The primitive is generated in response to Sec-SymmetricCryptomaterialHandle-HashedId8.request.

#### **9.3.8.2.4 Effect of receipt**

No behavior is specified.

#### **9.3.8.3 Sec-SymmetricCryptomaterialHandle-Delete.request**

##### **9.3.8.3.1 Function**

The primitive is used by a SDEE to request deletion of a Symmetric Cryptomaterial Handle (SCMH) and the associated cryptographic material.

##### **9.3.8.3.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-SymmetricCryptomaterialHandle-Delete.request (
    Symmetric Cryptomaterial Handle,
    )
```

Name	Type	Valid range	Description
<i>Symmetric Cryptomaterial Handle</i>	Integer	Any	A CMH as specified in 9.2.3 in any state

##### **9.3.8.3.3 When generated**

The primitive is generated as needed by SDEEs.

##### **9.3.8.3.4 Effect of receipt**

On receipt of this primitive the SDS deletes the linkage between the CMH and any cryptomaterial that it references. The SDS may also delete the cryptomaterial itself.

#### **9.3.8.4 Sec-SymmetricCryptomaterialHandle-Delete.confirm**

##### **9.3.8.4.1 Function**

The primitive confirms the operation of the Sec-SymmetricCryptomaterialHandle-Delete.request primitive.

##### **9.3.8.4.2 Semantics of the service primitive**

The primitive does not take parameters.

##### **9.3.8.4.3 When generated**

The primitive is generated in response to Sec-SymmetricCryptomaterialHandle-Delete.request.

#### **9.3.8.4.4 Effect of receipt**

No behavior is specified.

### **9.3.9 Sec-SignedData**

#### **9.3.9.1 Sec-SignedData.request**

##### **9.3.9.1.1 Function**

The primitive is used by a SDEE to request that the SDS signs data.

##### **9.3.9.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-SignedData.request (  
    Cryptomaterial Handle,  
    Data (optional),  
    Data Type (optional),  
    External Data Hash (optional),  
    External Data Hash Algorithm (optional),  
    PSID,  
    Set Generation Time,  
    Set Generation Location,  
    Expiry Time (optional),  
    Signer Identifier Type,  
    Signer Identifier Certificate Chain Length (optional),  
    Maximum Certificate Chain Length (optional),  
    Sign With Fast Verification,  
    EC Point Format,  
    Use Peer-to-Peer Cert Distribution,  
    SDEE ID (optional)  
)
```

Name	Type	Valid range	Description
<i>Cryptomaterial Handle</i>	Integer	Any integer	A CMH as specified in 9.2.2 in the Key and Certificate state, where the permissions of the certificate referenced by the CMH allow that certificate to generate a signature on the provided data.
<i>Data</i>	Octet string	Any	Used to fill in the <i>data</i> field of the SignedData-Payload.
<i>Data Type</i>	Enumerated	Ieee1609Dot2Data Raw	Present if and only if <i>Data</i> is present. Set to Raw if the contents of Data are to be encapsulated in an Ieee1609Dot2Data. Set to Ieee1609Dot2Data if the contents of Data are an already-encoded Ieee1609-Dot2Data.
<i>External Data Hash</i>	Octet string	Length 32	Used to fill in the <i>extDataHash</i> field of the Signed-DataPayload of ToBeSignedData payload.
<i>External Data Hash Algorithm</i>	Enumerated	sha256	If <i>External Data Hash</i> is non-empty then this field indicates the hash algorithm used to generate <i>External Data Hash</i> as specified in 5.3.1.
<i>PSID</i>	Integer	0...( $2^{32} - 1$ )	Used to fill in the <i>psid</i> field of the ToBeSignedData.
<i>Set Generation Time</i>	Boolean	True False	If True, the resulting ToBeSignedData contains the <i>generationTime</i> field.
<i>Set Generation Location</i>	Boolean	True False	If True, the resulting ToBeSignedData contains the <i>generationLocation</i> field.
<i>Expiry Time</i>	Time	Any time. If <i>Generation Time</i> is included, is later than or equal to <i>GenerationTime</i> .	If provided, the resulting ToBeSignedData contains the <i>expiryTime</i> field.
<i>Signer Identifier Type</i>	Enumerated	Certificate digest self	Sets the type of the <i>SignerIdentifier</i> within the SignedData.
<i>Signer Identifier Certificate Chain Length</i>	Integer or “Max”	1...256 -256...-1 “Max”	If Signer Identifier Type is “certificate”, sets the length of the certificate chain. If positive, includes that number of certificates from the chain. If negative with value -n, omits the top n certificates, starting with the root CA certificate, and includes the rest of the chain. If “Max”, includes the entire certificate chain back to the root certificate.
<i>Maximum Certificate Chain Length</i>	Integer	Any positive integer $\geq 2$	As specified in 5.1.2, the maximum number of certificates that may be in the complete certificate chain.
<i>Sign With Fast Verification</i>	Enumerated	Yes—uncompressed Yes—compressed No	If this is “Yes—uncompressed” or “Yes—compressed”, the confirm primitive returns data to enable fast verification. If this is “No”, the confirm primitive does not return this data, i.e., the type of R in the EccP256CurvePoint is set to x-only.
<i>EC Point Format</i>	Enumerated	Uncompressed Compressed	States whether elliptic curve points (public keys in explicit certificates and reconstruction values in implicit certificates) should be represented in compressed or uncompressed form as specified in EccP256CurvePoint.
<i>Use Peer-to-Peer Cert Distribution</i>	Boolean	True False	Whether or not to use peer-to-peer certificate distribution as specified in Clause 8. Specifically, whether or invoke SSME-Sec-OutgoingP2ped-Info.request within this primitive
<i>SDEE ID</i>	Integer		Provided if <i>Use Peer-to-Peer Cert Distribution</i> is true for use by SSME-Sec-OutgoingP2pcdInfo.request.

### **9.3.9.1.3 When generated**

The primitive is generated by a SDEE to request that the SDS signs data.

### **9.3.9.1.4 Effect of receipt**

On receipt of this primitive the SDS generates, if possible, a valid encoded Ieee1609Dot2Data of type SignedData containing the indicated payload.

If the parameter *Data* was provided and *Data Type* was Ieee1609Dot2Data, the result of the operation is an Ieee1609Dot2Data, whose content field contains a SignedData, in which tbsData.payload.data is equal to the parameter *Data*.

If the parameter *Data* was provided and *Data Type* was Raw, the result of the operation is an Ieee1609Dot2Data, whose content field contains a SignedData, in which tbsData.payload.data is in turn an Ieee1609Dot2Data whose content field contains unsecuredData which is the COER encoding of an octet string containing *Data*.

If the parameter *Use Peer-to-Peer Cert Distribution* is True, the SDS invokes SSME-Sec-OutgoingP2pcdInfo.request with parameters set as follows:

- *SDEE ID*: The SDEE ID parameter provided to Sec-SignedData.request.
- *Certificate*: The certificate indicated by the parameter *Cryptomaterial Handle* provided to Sec-SignedData.request.

If the corresponding SSME-Sec-OutgoingP2pcdInfo.confirm returns a p2pcdLearningRequest parameter, the SDS include that parameter in the HeaderInfo of the SignedData.

The result of the operation (the valid output on success, or an error code on failure) is returned via Sec-SignedData.confirm.

### **9.3.9.2 Sec-SignedData.confirm**

#### **9.3.9.2.1 Function**

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### **9.3.9.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-SignedData.confirm (
    Result Code,
    Signed Data (optional),
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Incorrect input No certificate provided No public key provided Not enough information to construct chain No trust anchor Chain too long for implementation Not cryptographically valid Unknown cryptographic validity Inconsistent permissions in chain Revoked Dubious Unsupported critical information fields Invalid encoding Current time before certificate validity period Current time after certificate validity period Expiry time before certificate validity period Expiry time after certificate validity period Invalid generation location Inconsistent permissions in certificate Incorrect requested certificate chain length for security profile Incorrect requested certificate chain length for implementation	The result of the signing operation as specified in 5.3.1
<i>Signed Data</i>	Octet string	An Ieee1609Dot2Data of type signedData	The SignedData, if it was created

### 9.3.9.2.3 When generated

The primitive is generated in response to Sec-SignedData.request. The parameters are set as follows. In the description below, “the input *XXX*” is shorthand for “the parameter *XXX* provided to the corresponding invocation of Sec-SignedData.request”.

- a) *Signed Data*:
  - 1) If the signing operation resulting from the Sec-SignedData.request succeeded, the *Signed Data* parameter contains the encoded signed data. This is an Ieee1609Dot2Data with:
    - i) content indicating type signedData.
    - ii) content.signedData.tbsData.payload.data containing:
      - i) If the input *Data Type* was Ieee1609Dot2Data, the input *Data* exactly as provided.
      - ii) If the input *Data Type* was Raw, an Ieee1609Dot2Data wth content indicating type unsecured and content.unsecuredData containing the input *Data*.
    - iii) A SignerIdentifier field containing the certificate or public key from the input *CMH*, with the signer identifier type and certificate chain (if appropriate) as indicated by Sec-SignedData.request.
    - iv) All other the fields set as indicated by Sec-SignedData.request.

- 2) If the signing operation resulting from the Sec-SignedData.request did not succeed, not present.
- b) *Result Code:*
- 1) *Result Code* is set as follows if only one error occurred when signing:
    - i) “Incorrect input” if neither the input *Data* nor the input *extDataHash* was provided.
    - ii) “No certificate provided” if Signer Identifier Type provided to Sec-SignedData.request was anything other than self and the CMH provided was not in Certificate and Key state.
    - iii) “No public key provided” if Signer Identifier Type provided to Sec-SignedData.request was self and the CMH provided was not in Key Pair Only state.
    - iv) “Not enough information to construct chain” if the SDS could not construct a chain to a trust anchor.
    - v) “No trust anchor” if the chain from the certificate does not end at a known trust anchor (see 5.1.2.1).
    - vi) “Chain too long for implementation” if the chain is longer than the implementation supports (see 5.1.2.3).
    - vii) “Not cryptographically valid” if any certificate in the chain fails to verify cryptographically (see 5.1.2.3).
    - viii) “Unknown cryptographic validity” if any certificate in the chain certificate has not been verified.
    - ix) “Inconsistent permissions in chain” if the permissions in the chain are inconsistent (see 5.1.2.4).
    - x) “Revoked” if any certificate in the chain has been revoked (see 5.1.3).
    - xi) “Dubious” if the revocation information relevant to any certificate in the chain is overdue (see 5.1.3.6).
    - xii) “Unsupported critical information fields” if the certificate or a certificate in its chain contains an unsupported critical information field (see 5.2.5).
    - xiii) “Invalid encoding” if the certificate or a certificate in its chain is not a valid encoding of the data structures in Clause 6.
    - xiv) “Current time before certificate validity period” if the time at which signing was requested is before the certificate’s start time.
    - xv) “Current time after certificate validity period” if the time at which signing was requested is after the certificate’s expiry time.
    - xvi) “Expiry time before certificate validity period” if Expiry Time is before the certificate’s start time.
    - xvii) “Expiry time after certificate validity period” if Expiry Time is after the certificate’s expiry time.
    - xviii) “Invalid generation location” if the region field was present in the signing ToBeSignedCertificate and the current location is outside that region.
    - xix) “Inconsistent permissions in certificate” if Cryptomaterial Handle does not contain a (PSID, SSP) pair equal to (PSID, Service Specific Permissions).
    - xx) “Incorrect requested certificate chain length for security profile” if the length of the certificate chain from the signing certificate to the root is greater than Maximum Certificate Chain Length.

- xxi) “Incorrect requested certificate chain length for implementation” if the length of the certificate chain from the signing certificate to the root is greater than the maximum length supported by the implementation.
- 2) If the signing operation fails for more than one of the reasons above, Result Code takes a value indicating one of the reasons.
- 3) Result Code is set to “success” if none of the abovementioned conditions hold.

#### **9.3.9.2.4 Effect of receipt**

No behavior is specified.

### **9.3.10 Sec-EncryptedData**

#### **9.3.10.1 Sec-EncryptedData.request**

##### **9.3.10.1.1 Function**

The primitive is used by a SDEE to request that the SDS encrypts data.

##### **9.3.10.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-EncryptedData.request (
    Data,
    Data Type,
    Data Encryption Key Type,
    Symmetric CMHs (optional),
    Recipient Certificates (optional),
    Signed Data Recipient Info (optional),
    Response Encryption Key (optional),
    EC Point Format
)
```

If none of the parameters *Recipient Certificates*, *Symmetric CMHs*, *Signed Data Recipient Info*, *Response Encryption Key* are provided, the corresponding confirm primitive returns an error.

Name	Type	Valid range	Description
<i>Data</i>	Octet string	An octet string Ieee1609Dot2Data	The data to be encrypted
<i>Data Type</i>	Enumerated	Ieee1609Dot2Data Raw	Set to “Raw” if the contents of Data are an octet string, not wrapped in an Ieee1609Dot2Data. Set to “Ieee1609Dot2Data” if the contents of Data are an already-encoded Ieee1609Dot2Data.
<i>Data Encryption Key Type</i>	Enumerated	“static” “ephemeral”	Whether the data is to be encrypted with a static data encryption key as in 5.3.4.2, or an ephemeral data encryption key as in 5.3.4.1.
<i>Symmetric CMH</i>	Array of Symmetric Cryptomaterial Handles		A Symmetric CMH as specified in 9.2.3. If <i>Data Encryption Key Type</i> is “static”, this parameter is present and has a single entry. Otherwise, this may or may not be present.
<i>Recipient Certificates</i>	Array of certificates	Any array of valid certificates which contain encryption keys	One certificate for each recipient. Not present if <i>Data Encryption Key Type</i> is “static”. Optionally present if <i>Data Encryption Key Type</i> is “ephemeral”.
<i>Signed Data Recipient Info</i>	Tuple of (public key, 32-byte octet string)	The public key is for an encryption algorithm	The 32-byte octet string is the hash of the signed data from which the public key was obtained as specified in 6.3.31. Not present if <i>Data Encryption Key Type</i> is “static”. Optionally present if <i>Data Encryption Key Type</i> is “ephemeral”.
<i>Response Encryption Key</i>	Public key	HashedId8 of the response encryption key	Public key is for an encryption algorithm. Not present if <i>Data Encryption Key Type</i> is “static”. Optionally present if <i>Data Encryption Key Type</i> is “ephemeral”.
<i>EC Point Format</i>	Enumerated	Compressed Uncompressed	The format of the elliptic curve points included in the <i>Recipient Infos</i>

### 9.3.10.1.3 When generated

The primitive is generated by a SDEE to request that the SDS encrypts data.

### 9.3.10.1.4 Effect of receipt

On receipt of this primitive the SDS attempts to encrypt the data for the specified recipients as specified in 5.3.4. If *Data Type* is raw, the SDS first encapsulates *Data* in an Ieee1609Dot2Data by setting it as the payload of an Ieee1609Dot2Data of type unsecuredData.

## 9.3.10.2 Sec-EncryptedData.confirm

### 9.3.10.2.1 Function

This primitive returns the values calculated in the processing specified for the corresponding request primitive.

### 9.3.10.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-EncryptedData.confirm (
    Result Code,
    Encrypted Data (optional),
    Failed Certificates (optional)
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Incorrect inputs Fail on some certificates Fail on all certificates	The result of the encryption operation as specified in 5.3.4
<i>Encrypted Data</i>	Octet string	An IEEE1609.2Dof2Data of type encryptedData	The encrypted data, if it was created
<i>Failed Certificates</i>	Certificate array	Any	Any certificates on which encryption failed

### 9.3.10.2.3 When generated

The primitive is generated in response to Sec-EncryptedData.request. The parameters are set as follows. In the description below, “the input *XXX*” is shorthand for “the parameter *XXX* provided to the corresponding invocation of Sec-EncryptedData.request”.

- a) *Result Code* is set as follows:
  - 1) *Result Code* is set to “fail on some certificates” if any of the following hold:
    - i) At least one of the certificates passed to Sec-EncryptedData.request is not known to the SSME, i.e., a query of SSME-CertificateInfo.request results in a *Result Code* from SSME-CertificateInfo.confirm other than “found”.
    - ii) At least one of the certificates passed to Sec-EncryptedData.request does not contain an encryption key.
    - iii) The public key algorithm for the encryption key in at least one of the certificates is not a known value.
    - iv) The symmetric encryption algorithm associated with the encryption key in at least one of the certificates is not supported by this implementation.
  - 2) *Result Code* is set to “fail on all certificates” if none of the certificates could be encrypted to for one of the reasons above.
  - 3) *Result Code* is set to “incorrect inputs” if none of the parameters *Recipient Certificates*, *Symmetric CMH*, *Signed Data Recipient Info*, *Response Encryption Key* were provided to the corresponding request primitive.
  - 4) *Result code* is set to “success” if all of the recipients could be encrypted to.
- b) *Failed Certificates* is empty if *Result Code* is “success”, and otherwise contains an array of the certificates for which encryption failed.
- c) *Encrypted Data* is empty if *Result Code* is “fail on all certificates”, and otherwise contains the encoded encrypted data as specified in 5.3.4.

- 1) If the input *Data Type* was Ieee1609Dot2Data, the plaintext P is the input *Data* exactly as provided.
- 2) If the input *Data Type* was Raw, the plaintext P is an Ieee1609Dot2Data wth content indicating type unsecured and content.unsecuredData containing the input *Data*.

#### **9.3.10.2.4 Effect of receipt**

No behavior is specified.

### **9.3.11 Sec-SecureDataPreprocessing**

#### **9.3.11.1 Sec-SecureDataPreprocessing.request**

##### **9.3.11.1.1 Function**

The primitive is used by a SDEE to request that the SDS performs the preprocessing on secure data.

##### **9.3.11.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-SecureDataPreprocessing.request(
    Data,
    SDEE ID (optional),
    PSID,
    Use P2PCD
)
```

Name	Type	Valid range	Description
<i>Data</i>	Octet string	An Ieee1609-Dot2Data	The data to be processed
<i>SDEE ID</i>	SDEE ID		The SDEE ID of the invoking SDEE
<i>PSID</i>	PSID		The PSID derived from context (see 5.2.3.3.2)
<i>Use P2PCD</i>	Boolean	True False	Whether or not this should initiate a P2PCD learning request process.

##### **9.3.11.1.3 When generated**

The primitive is generated by a SDEE to request that the SDS performs preprocessing on an Ieee1609-Dot2Data.

##### **9.3.11.1.4 Effect of receipt**

On receipt of this primitive, the SDS takes the following actions:

- a) Determine the type of the input Ieee1609Dot2Data.

- b) If the input is of type `signedData`:
  - 1) Determine the SSP associated with the input PSID in the signer's certificate, if it is available.
  - 2) If `Use P2PCD` is True, invoke `SSME-Sec-IncomingP2pcdInfo.request` with parameters equal to the input SDEE ID and the certificate and P2PCD request from the `SignedData`, if any. No action is specified to be taken based on the parameters of the corresponding `SSME-Sec-IncomingP2pcdInfo.confirm`.
  - 3) If the `SignerIdentifier` is of type `certificate`, add the certificate(s) to the SSME by invoking `SSME-AddCertificate.request`.

NOTE—Previous versions of this standard had an equivalent to this function returning various fields from the `Ieee1609Dot2Data`. In this version of the standard the specification assumes that the SDEE can parse an `Ieee1609Dot2Data` and all that needs to be specified are actions that the SDEE cannot carry out by itself.

### **9.3.11.2 Sec-SecureDataPreprocessing.confirm**

#### **9.3.11.2.1 Function**

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### **9.3.11.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-SecureDataPreprocessing.confirm(  
    Result Code,  
    Content Type (optional),  
    Service Specific Permissions (optional),  
    Assurance Level (optional),  
    Earliest Next CRL Time  
)
```

Name	Type	Valid range	Description	When included
<i>Result Code</i>	Enumerated	Success Invalid input Unknown certificate Inconsistent PSID	The result of the data extraction operation.	
<i>Content Type</i>	Enumerated	Unsecured Encrypted Signed	The type of the IEEE1609-Dot2Data passed in the request.	Included if <i>Result Code</i> is “success”.
<i>Service Specific Permissions</i>	Octet string	An octet string of length 0–32 octets	The SSP from the certificate that validates the signed data.	Included if <i>Result Code</i> is “success” and the certificate included a SSP with the indicated PSID.
<i>Assurance Level</i>	Subject Assurance as specified in 6.4.27		The assurance level from the certificate that validates the signed data.	Included if <i>Result Code</i> is “success” and the certificate included an assurance level.
<i>Earliest Next CRL Time</i>	Time	Any valid time	The earliest nextCrl time value for any certificate in the chain for a signed SPDU.	Included if <i>Data</i> was of type signed and <i>Result Code</i> is “success”.

### 9.3.11.2.3 When generated

The primitive is generated in response to Sec-SecureDataPreprocessing.request. The parameters are set as follows. In the description below, “the input XXX” is shorthand for “the parameter XXX provided to the corresponding invocation of Sec-SecureDataPreprocessing.request”.

- a) *Result Code* is set as follows:
  - 1) “Invalid input” if the input *Data* couldn’t be parsed.
  - 2) “Unknown certificate” if the following all hold:
    - i) The input *Data* was of type Signed.
    - ii) The SignerIdentifier in the SignedData of the input *Data* was of type digest.
    - iii) The corresponding certificate to the SignerIdentifier is not known to the SSME as defined in 4.3.
  - 3) “Inconsistent PSID” if the input PSID does not appear in the certificate.
  - 4) “Success” if none of the above conditions hold.
- b) *Content Type* is set only if Result Code is success and indicates the type of content contained in the input *Data*.
- c) *Service Specific Permissions* is set only if Result Code is success and Content Type is signed. It indicates the Service Specific Permissions of the certificate that signed the input *Data*.
- d) *Assurance Level* is set only if Result Code is success and Content Type is signed. It contains the SubjectAssurance from the ToBeSignedCertificate of the certificate that signed the input *Data*. If there was no SubjectAssurance field, this is omitted.
- e) *Earliest Next CRL Time* is set only if Result Code is success and Content Type is signed. It indicates the earliest nextCrl value from any certificate in the chain that signed the input *Data* as specified in 5.1.3.6.

### 9.3.11.2.4 Effect of receipt

None specified.

## 9.3.12 Sec-SignedDataVerification

### 9.3.12.1 Sec-SignedDataVerification.request

#### 9.3.12.1.1 Function

The primitive is used by a SDEE to request that the SDS verifies signed data.

#### 9.3.12.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-SignedDataVerification.request (  
    SDEE ID,  
    PSID,  
    Content Type,  
    Signed Data,  
    External Data Hash (optional),  
    External Data Hash Algorithm (optional),  
    Maximum Certificate Chain Length (optional),  
    Relevance: Replay,  
    Relevance: Generation Time in Past,  
    Validity Period (optional),  
    Relevance: Generation Time in Future,  
    Acceptable Future Data Period (optional),  
    Generation Time (optional),  
    Relevance: Expiry Time,  
    Expiry Time (optional),  
    Consistency: Generation Location (optional),  
    Relevance: Generation Location Distance,  
    Validity Distance (optional),  
    Generation Location (optional),  
    Overdue CRL Tolerance (optional),  
    Relevance: Expired Certificate  
)
```

Name	Type	Valid range	Description
SDEE ID	Integer	Any	The SDEE ID of the SDEE.
PSID	PSID	Any	The PSID derived from context (see 5.2.3.3.2).
Content Type	Enumerated	Signed Signed partial payload Signed external payload	The type of the SignedData.
Signed Data	Ieee1609Dot2-Data	An Ieee1609Dot2-Data of type signedData	The signed data.
External Data Hash	Octet string	Octet string of length 32	The hash of external data, to be checked against the extDataHash field in the signedData.
External Data Hash Algorithm	HashAlgorithm	sha256	If External Data Hash is non-empty then this field indicates the hash algorithm used to generate External Data Hash as specified in 5.3.1.
Maximum Certificate Chain Length	Integer	Any integer $\geq 2$	The maximum length the certificate chain may have as specified in 5.1.2.
Relevance: Replay	Boolean	True False	If “True”, the SDS carries out replay detection as specified in 5.2.4.2.6 using SSME-Sec-ReplayDetection.request. In this case, at least one of Relevance: Expiry Time and Relevance: Generation Time in Past should be set to “True”, and Validity Period shall be provided.
Generation Time	Time	Any	The generation time to use in the security processing. This field is required if the HeaderInfo field in Signed Data does not contain generationTime.
Relevance: Generation Time in Past	Boolean	True False	If “True”, the SDS rejects too-old SPDUs as specified in 5.2.4.2.2.
Validity Period	Time period	Any time period	The period after the generation time for which the content is of interest to the recipient. Provided if Relevance: Replay or Relevance: Generation Time in Past is “True”.
Relevance: Generation Time in Future	Boolean	True False	If “True”, the SDS rejects future SPDUs as specified in 5.2.4.2.3.
Acceptable Future Data Period	Time	Any positive time value	Used in conjunction with Rejection Threshold for Generation Time in Future to determine if data should be rejected because its generation time is in the future as specified in 5.2.4.2.3. Provided if Relevance: Generation Time in Future is “True”.
Relevance: Expiry Time	Boolean	True False	If “True”, the SDS rejects SPDUs if the local time is after the expiry time as specified in 5.2.4.2.4.
Expiry Time	Time	Any	The expiry time to use in the security processing. This field is required if Relevance: Expiry Time is “True” and the HeaderInfo field in Signed Data does not contain expiryTime.
Consistency: Generation Location	Boolean	True False	If True, the SDS checks that the generation location is inside the validity region of the certificate.
Relevance: Generation Location Distance	Boolean	True False	If True, the SDS performs relevance checks based on the generation location as specified in 5.2.4.2.5.
Validity Distance	Distance	Any positive value	The maximum allowed distance between the recipient and the generation location. This is provided if Reject Too Distant Messages is “True”.

Name	Type	Valid range	Description
<i>Generation Location</i>	A 3D location or “none”	Any value indicating a position on or near the surface of the earth	The generation location. This field is required if the HeaderInfo field in <i>Signed Data</i> does not contain generationLocation.
<i>Overdue CRL Tolerance</i>	Time or “any”	Any time period (e.g., minutes, days, years), or “Any”	If a CRL relevant to a certificate in the sending chain was due to be issued more than <i>Overdue CRL Tolerance</i> time ago, and has not been received, the chain is rejected.
<i>Relevance: Expired Certificate</i>	Boolean	True False	If True, the SDS checks that none of the certificates in the chain that signed <i>Signed Data</i> have expired per 5.2.4.2.7.

### 9.3.12.1.3 When generated

The primitive is generated by a SDEE to request that the SDS verifies signed data that was obtained from a previous Sec-SecureDataPreprocessing.request.

### 9.3.12.1.4 Effect of receipt

Upon receipt of this primitive, the SDS determines whether the data is valid by the criteria of 5.2. If so requested by the *Relevance: Replay* parameter, the SDS invoke SSME-Sec-ReplayDetection.request, with parameters:

- *SDEE ID* = the *SDEE ID* parameter to this primitive
- *Signed Data* = the *Data* parameter to this primitive
- *Discard Time* = either the generation time + *Validity Period* or the expiry time, whichever comes earlier

## 9.3.12.2 Sec-SignedDataVerification.confirm

### 9.3.12.2.1 Function

The primitive returns the result of the the corresponding request primitive.

### 9.3.12.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
Sec-SignedDataVerification.confirm (
    Result Code,
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success  Inconsistent input parameters  SPDU-Parsing: Invalid Input SPDU-Parsing: Unsupported critical information field SPDU-Parsing: Certificate not found SPDU-Parsing: Generation time not available SPDU-Parsing: Generation location not available  SPDU-Certificate-Chain: Not enough information to construct chain SPDU-Certificate-Chain: Chain ended at untrusted root SPDU-Certificate-Chain: Chain was too long for implementation SPDU-Certificate-Chain: Certificate revoked SPDU-Certificate-Chain: Overdue CRL SPDU-Certificate-Chain: Inconsistent expiry times, SPDU-Certificate-Chain: Inconsistent start times SPDU-Certificate-Chain: Inconsistent chain permissions  SPDU-Crypto: Verification failure  SPDU-Consistency: Future certificate at generation time SPDU-Consistency: Expired certificate at generation time SPDU-Consistency: Expiry date too early SPDU-Consistency: Expiry date too late SPDU-Consistency: Generation location outside validity region SPDU-Consistency: No generation location SPDU-Consistency: Unauthorized PSID  SPDU-Internal-Consistency: Expiry time before generation time SPDU-Internal-Consistency: extDataHash doesn't match SPDU-Internal-Consistency: no extDataHash provided SPDU-Internal-Consistency: no extDataHash present  SPDU-Local-Consistency: PSIDs don't match SPDU-Local-Consistency: Chain was too long for SDEE  SPDU-Relevance: Generation Time too far in past SPDU-Relevance: Generation Time too far in future SPDU-Relevance: Expiry Time in past SPDU-Relevance: Generation Location too distant SPDU-Relevance: Replayed SPDU SPDU-Relevance: Certificate expired	The result of the validation operation

### 9.3.12.2.3 When generated

The primitive is generated in response to Sec-SignedDataVerification.request. This subclause specifies how the field Result Code is set. In the description below, “the input XXX” is shorthand for “the parameter XXX provided to the corresponding invocation of Sec-SignedData.request”. The term “the SPDU YYY was [not] available”, for YYY taking the values Generation Time, Expiry Time, Generation Location means that YYY was [not] either encoded in the input *Signed Data* or provided via the input *YYY*.

Result Code is set as follows:

- a) “Success” if the signed SPDU passed all consistency (as specified in 5.2.3) and relevance (as specified in 5.2.4) checks requested.
- b) “Inconsistent input parameters” if:
  - 1) The input *SignedData* contained a generation time and the input *Generation Time* was provided.
  - 2) The input *SignedData* contained an expiry time and the input *Expiry Time* was provided.
  - 3) The input *SignedData* contained a generation location and the input *Generation Location* was provided.
- c) “SPDU-Parsing: Invalid Input” if the SPDU could not be parsed or is not of type SignedData.
- d) “SPDU-Parsing: Unsupported critical information field” if the SPDU contained a critical information field as identified in Clause 6 which is not supported by the implementation.
- e) “SPDU-Parsing: Certificate not found” if the SPDU’s SignerIdentifier is of type *digest* and the corresponding certificate is not known to the SSME as defined in 4.3.
- f) “SPDU-Parsing: Generation time not available” if the SPDU *Generation Time* was not available as defined above.
- g) “SPDU-Parsing: Generation location not available” if the input *Relevance: Generation Location Distance* was true, and one of the following holds:
  - 1) The SPDU *Generation Location* was not available as defined above, or
  - 2) The encoding of the SPDU *Generation Location* uses one of the “not available” values specified in 6.3.13, 6.3.14, or 6.3.15 to indicate that generation location was not available at the time the signature was generated.
- h) “SPDU-Certificate-Chain: Not enough information to construct chain” if one of the certificates in the chain had an issuer value that is not known to the SSME as defined in 4.3.
- i) “SPDU-Certificate-Chain: Chain ended at untrusted root” if the certificate chain can be constructed to a root, i.e., to a certificate with *issuer* indicating *self*, but that root is not trusted.
- j) “SPDU-Certificate-Chain: Chain was too long for implementation” if the certificate chain is longer than is supported by the implementation as specified in 5.1.2.2.
- k) “SPDU-Certificate-Chain: Certificate revoked” if any certificate in the chain is revoked as specified in 5.1.3.
- l) “SPDU-Certificate-Chain: Overdue CRL” if revocation information for any certificate in the chain is overdue by more than the input *Overdue CRL Tolerance* as specified in 5.1.3.6.
- m) “SPDU-Certificate-Chain: Inconsistent expiry times” if for some pair of certificates in the chain the subordinate certificate’s expiry time was after the issuing certificate’s expiry time as specified in 5.1.2.3.

- n) “SPDU-Certificate-Chain: Inconsistent start times” if for some pair of certificates in the chain the subordinate certificate’s start validity time was before the issuing certificate’s start validity time as specified in 5.1.2.3.
- o) “SPDU-Certificate-Chain: Inconsistent chain permissions” if for some pair of certificates in the chain the subordinate certificate’s permissions are not consistent with the issuing certificate’s permissions as defined in 5.2.3.2.2.
- p) “SPDU-Crypto: Verification failure” if the signature on the SPDU or on any explicit certificate in the chain do not pass cryptographic verification as defined in 5.3.1.
- q) “SPDU-Consistency: Future certificate at generation time” if the SPDU generation time is before the start validity time of the signing certificate as specified in 5.1.2.3.
- r) “SPDU-Consistency: Expired certificate at generation time” if the SPDU generation time is after the expiry time of the signing certificate as specified in 5.1.2.3.
- s) “SPDU-Consistency: Expiry date too early” if the SPDU expiry time was provided, and that time is before the start validity time of the signing certificate.
- t) “SPDU-Consistency: Expiry date too late” if the SPDU expiry time was provided, and that time is after the expiry time of the signing certificate.
- u) “SPDU-Consistency: Generation location outside validity region” if the input *Consistency: Generation Location* was True, the SPDU generation location was provided, and the SPDU generation location is outside the validity region of the signing certificate as specified in 5.2.3.2.2.
- v) “SPDU-Consistency: No generation location” if the input *Consistency: Generation Location* was True and one of the following holds:
  - 1) The SPDU *Generation Location* was not available as defined above, or
  - 2) The encoding of the SPDU *Generation Location* uses one of the “not available” values specified in 6.3.13, 6.3.14, or 6.3.15 to indicate that generation location was not available at the time the signature was generated.
- w) “SPDU-Consistency: Unauthorized PSID” if the input *PSID* does not appear in the signing certificate as specified in 5.2.3.3.2.
- x) “SPDU-Internal-Consistency: Expiry time before generation time” if the SPDU expiry time is provided and is before the SPDU generation time.
- y) “SPDU-Internal-Consistency: extDataHash doesn’t match” if the input *Signed Data* has a Signed-DataPayload containing an *extDataHash*, the input *External Data Hash* was provided, and the input *External Data Hash* hash does not match the *extDataHash* in the *Signed Data*.
- z) “SPDU-Internal-Consistency: no extDataHash provided” if the input *Signed Data* has a Signed-DataPayload containing an *extDataHash* and the input *External Data Hash* was not provided.
- aa) “SPDU-Internal-Consistency: no extDataHash present” if the the input External Data Hash was provided but the input *Signed Data* has a SignedDataPayload that does not contain an *extDataHash*.
- bb) “SPDU-Local-Consistency: PSIDs don’t match” if the input *PSID* is not the same as the *psid* field in the HeaderInfo of the input *Signed Data*.
- cc) “SPDU-Local-Consistency: Chain was too long for SDEE” if the input *Maximum Certificate Chain Length* was provided and if the length of the signing certificate’s chain is greater than *Maximum Certificate Chain Length* as specified in 5.2.3.3.1.
- dd) “SPDU-Relevance: Generation Time too far in past” if the input *Relevance: Generation Time in Past* is True and the generation time is far in the past by the criteria specified in 5.2.4.2.2.
- ee) “SPDU-Relevance: Generation Time too far in future” if the input *Relevance: Generation Time in Future* is True and the generation time is too far in the future by the criteria specified in 5.2.4.2.3.

- ff) “SPDU-Relevance: Expiry Time in past” if the input *Relevance: Expiry Time* is True and the expiry time is calculated to be too far in the past by the criteria specified in 5.2.4.2.4.
- gg) “SPDU-Relevance: Generation Location too distant” if the input *Relevance: Generation Location Disatnce* is True and the generation location is calculated to be too distant by the criteria specified in 5.2.4.2.5.
- hh) “SPDU-Relevance: Replicated SPDU” if the input *Relevance: Replay* is True and the Signed Data is an exact duplicate of a Signed Data previously submitted to Sec-SignedDataVerification.request with the same value for the input *SDEE ID*. This may be implemented via SSME-Sec-ReplayDetection.request, SSME-Sec-ReplayDetection.confirm.
- ii) “SPDU-Relevance: Certificate Expired” if the input *Relevance: Expired Certificate* is True and any certificate in the chain that signed *Signed Data* has expired.
- jj) If the input Signed Data fails more than one of the validity conditions, Result Code takes a value indicating one of the reasons.

#### **9.3.12.2.4 Effect of receipt**

No behavior is specified.

### **9.3.13 Sec-EncryptedDataDecryption**

#### **9.3.13.1 Sec-EncryptedDataDecryption.request**

##### **9.3.13.1.1 Function**

This primitive is used by a SDEE to request the SDS to decrypt encrypted data with the provided Cryptomaterial Handle.

##### **9.3.13.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-EncryptedDataDecryption.request (
    Data,
    Cryptomaterial Handle,
    Signed Data Recipient Info (optional)
)
```

Name	Type	Valid range	Description
<i>Data</i>	Octet string	An Ieee1609Dot2-Data of type encryptedData	The encrypted data to be decrypted.
<i>Cryptomaterial Handle</i>	Integer	Any	CMH in <i>Key Pair Only</i> or <i>Key and Certificate</i> state, where the private key is for a decryption algorithm. Alternatively, a Symmetric Crypto Material Handle.
<i>Signed Data Recipient Info</i>	32-byte octet string	Any	In the case where the data being decrypted was encrypted with a key obtained from a SignedData within an Ieee1609-Dot2Data, the hash of that Ieee1609Dot2Data as specified in 5.3.5. Otherwise, omitted.  The decrypter can tell which key was used to encrypt an encrypted SPDU by inspecting the RecipientInfo fields within the EncryptedData.

### 9.3.13.1.3 When generated

The primitive is generated as needed within the WAVE Security Services or by other entities or processes.

### 9.3.13.1.4 Effect of receipt

On receipt, the SDS determines whether the input *Cryptomaterial Handle* corresponds to any of the RecipientInfo fields in *Data*. *Cryptomaterial Handle* is determined to correspond to a RecipientInfo field using the following criteria:

- a) Cryptomaterial Handle corresponds to a RecipientInfo *r* of type pskRecipInfo or symmRecipInfo if both of these conditions hold:
  - 1) *Cryptomaterial Handle* is a Symmetric Cryptomaterial Handle.
  - 2) The HashedId8 value contained in *r* is equal to the value returned on invoking Sec-SymmetricCryptomaterialHandle-HashedId8.request with parameter Cryptomaterial Handle.
- b) Cryptomaterial Handle corresponds to a RecipientInfo *r* of type certRecipInfo if both of these conditions hold:
  - 1) Cryptomaterial Handle is in state Certificate and Key.
  - 2) The HashedId8 of the certificate referenced by Cryptomaterial Handle is equal to the recipientId field in the PKRecipientInfo.
- c) Cryptomaterial Handle corresponds to a RecipientInfo *r* of type signedDataRecipInfo if both of these conditions hold:
  - 1) Cryptomaterial Handle is in state Key Pair Only.
  - 2) The parameter Signed Data Recipient Info was provided and the low-order eight bytes of that parameter are equal to the recipientId field in the PKRecipientInfo.
- d) Cryptomaterial Handle corresponds to a RecipientInfo *r* of type rekRecipInfo if both of these conditions hold:
  - 1) Cryptomaterial Handle is in state Key Pair Only.
  - 2) The HashedId8 of the public key referenced by Cryptomaterial Handle is equal to the recipientId field in the PKRecipientInfo.

If *Cryptomaterial Handle* corresponds to one of the entries in EncryptedData.recipients, the SDS attempts to decrypt the encrypted data with the cryptomaterial referenced by *Cryptomaterial Handle*.

### **9.3.13.2 Sec-EncryptedDataDecryption.confirm**

#### **9.3.13.2.1 Function**

The primitive returns the result of the the corresponding request primitive.

#### **9.3.13.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
Sec-EncryptedDataDecryption.confirm (
    Result Code,
    Data
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success No decryption key available Unsupported critical information field Couldn't decrypt key Couldn't decrypt data Invalid form for plaintext	The result of the decryption operation.
<i>Data</i>	Octet string	An Ieee1609Dot2Data of any type	If <i>Result Code</i> is “success”, the decrypted data. Otherwise, undefined.

#### **9.3.13.2.3 When generated**

The primitive is generated in response to Sec-EncryptedDataDecryption.request. This subclause specifies how the field Result Code is set. In the description below, “the input XXX” is shorthand for “the parameter XXX provided to the corresponding invocation of Sec-EncryptedDataDecryption.request”.

- a) *Result Code* is set as follows:
  - 1) “Invalid input” if the input *Data* couldn’t be parsed.
  - 2) “No decryption key available” if the input *Cryptomaterial Handle* did not correspond to any of the RecipientInfo fields in the EncryptedData by the criteria specified under Sec-EncryptedDataDecryption.request.
  - 3) “Unsupported critical information field” if the recipients field of the EncryptedData contains more entries than the implementation supports.
  - 4) “Couldn’t decrypt key” if the attempt to decrypt the data encryption key material in the RecipientInfo failed.
  - 5) “Couldn’t decrypt data” if the attempt to decrypt the data in the SymmetricCiphertext, using decryption with AES-CCM as specified in 5.3.6, failed.
  - 6) “Invalid form for plaintext” if the decrypted plaintext did not have the form of a valid Ieee1609Dot2Data.
  - 7) “Success” if none of the above conditions hold.
- b) *Data* is set only if Result Code is success and contains the decrypted data.

### 9.3.13.2.4 Effect of receipt

None specified.

## 9.4 SSME SAP

### 9.4.1 SSME-CertificateInfo

#### 9.4.1.1 SSME-CertificateInfo.request

##### 9.4.1.1.1 Function

This primitive is used by a process to query the SSME for information about the contents, revocation status, and inherited permissions of a certificate.

##### 9.4.1.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-CertificateInfo.request (
    Identifier Type,
    Identifier
)
```

Name	Type	Valid range	Description
<i>Identifier Type</i>	Enumerated	Certificate HashedId8 HashedId10	Indicates the type of input data used to identify the certificate
<i>Identifier</i>	Octet string	Any	The encoded certificate, HashedId8, or HashedId10 identifying the certificate in question

##### 9.4.1.1.3 When generated

The primitive is generated as needed within the WAVE Security Services or by other entities or processes.

##### 9.4.1.1.4 Effect of receipt

On receipt, the SSME determines whether the certificate indicated by *Identifier* is known to it. If so, it retrieves the information stored about that certificate as specified in 4.3. It returns the stored information, or an indication that the certificate is unknown, or an indication that *Identifier* identifies two or more certificates known to the SSME (which is possible if *Identifier Type* is HashedId8 or HashedId10), via SSME-CertificateInfo.confirm.

#### **9.4.1.2 SSME-CertificateInfo.confirm**

##### **9.4.1.2.1 Function**

This primitive returns the certificate identified in the corresponding request, if found.

##### **9.4.1.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-CertificateInfo.confirm (
    Result Code
    Certificate Data,
    Geographic Scope,
    Last Received CRL Time,
    Next Expected CRL Time,
    Trust Anchor
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Certificate not found Multiple certificates identified Not yet verified Verified and trusted No trust anchor Chain too long for implementation Not cryptographically valid Inconsistent permissions in chain Revoked Dubious Unsupported critical information fields Invalid encoding	The result of the request made in the corresponding request primitive.
<i>Certificate Data</i>	Array of octet strings	One or more validly encoded Certificates	The certificate or certificates indicated by the corresponding request primitive.
<i>Geographic Scope</i>	An array of geographic regions		The geographic region relevant to the certificate: the geographic region contained in the certificate or, if the certificate does not contain a geographic region, the geographic region from the first certificate above it in the chain to have one.
<i>Last Received CRL Time</i>	Date or “none”	Any date in the past	If available, the generation time of the last CRL received that would have contained the certificate if it had been revoked. Otherwise, “none”.
<i>Next Expected CRL Time</i>	Date or “unknown”	Any date after Last Received CRL Time	If available, the next time a CRL is going to be generated that could contain the certificate. This time may be in the past or the future. Otherwise, “unknown”.
<i>Trust Anchor</i>	Boolean	True, False	Whether or not the certificate is a trust anchor.
<i>Verified</i>	Boolean	True, False	True indicates that the cryptographic validity of the certificate is known to the SSME, either because the caller indicated that the certificate had been verified when it was added via SSME-AddCertificate.request, or because the certificate was verified later via SSME-VerifyCertificate.request. False indicates that the cryptographic validity is not established, i.e., that Verified was set to “False” on every invocation of SSME-AddCertificate.request with that certificate and that SSME-VerifyCertificate.request has not been subsequently invoked.

#### 9.4.1.2.3 When generated

The primitive is generated in response to SSME-CertificateInfo.request. The parameters are set as follows. In the description below, “the input XXX” is shorthand for “the parameter XXX provided to the corresponding invocation of SSME-CertificateInfo.request”.

- a) *Result Code* is set as follows:
    - 1) If the input *Identifier Type* was HashedId8 or HashedId10:
      - i) If the input *Identifier* did not correspond to any certificate known to the SSME, *Result Code* is set to “certificate not found”.
      - ii) If the input *Identifier* was the HashedId8 or HashedId10 of more than one certificate known to the SSME, *Result Code* is set to “multiple certificates identified” and *Certificate Data* contains all the certificates that correspond to the input *Identifier*.
    - 2) If the certificate has not yet been verified, i.e., if *Verified* was set to “False” on every invocation of *SSME-AddCertificate.request* with that certificate and that *SSME-VerifyCertificate.request* has not been subsequently invoked, *Result Code* is set equal to “Not yet verified”.
    - 3) Otherwise the cryptographic validity of the certificate is known to the SSME, either because the caller indicated that the certificate had been verified when it was added via *SSME-AddCertificate.request*, or because the certificate was verified later via *SSME-VerifyCertificate.request*. In this case, if one of the following error conditions holds, *Result Code* is set to indicate that error condition as follows:
      - i) “No trust anchor” if the chain from the certificate does not end at a known trust anchor (see 5.1.2.1).
      - ii) “Chain too long for implementation” if the chain is longer than the implementation supports (see 5.1.2.3).
      - iii) “Not cryptographically valid” if any certificate in the chain fails to verify cryptographically (see 5.1.2.3).
      - iv) “Unknown cryptographic validity” if the certificate has not been verified.
      - v) “Inconsistent permissions in chain” if the permissions in the chain are inconsistent (see 5.1.2.4).
      - vi) “Revoked” if the certificate has been revoked (see 5.1.3).
      - vii) “Dubious” if the revocation information relevant to the certificate is overdue (see 5.1.3.6).
      - viii) “Unsupported critical information fields” if the certificate or a certificate in its chain contains an unsupported critical information field (see 5.2.5).
      - ix) “Invalid encoding” if the certificate or a certificate in its chain is not a valid encoding of the data structures in Clause 6.
  - 4) If more than one of the error conditions in step 3) holds, *Result Code* is set to indicate any one of the applicable error conditions.
  - 5) If none of the above conditions apply, *Result Code* is set to “verified and trusted”.
- b) *Certificate Data* is set equal to the encoded certificate indicated by the input *Identifier*. If *Identifier* identifies more than one certificate, *Certificate Data* is an array of all known encoded certificates that correspond to the input *Identifier*.
- c) *Last Received CRL Time* is the last time relevant revocation information was received, if any (see 5.1.3).
- d) *Next Expected CRL Time* is the next time revocation information is expected to be received, if any (see 5.1.3). This time may be in the past.
- e) *Trust Anchor* indicates whether or not the certificate has been marked as a trust anchor within the SSME.

#### **9.4.1.2.4 Effect of receipt**

None specified.

### **9.4.2 SSME-AddTrustAnchor**

#### **9.4.2.1 SSME-AddTrustAnchor.request**

##### **9.4.2.1.1 Function**

The primitive is used by a process to add a trust anchor to the SSME's store of trust anchors.

##### **9.4.2.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddTrustAnchor.request (
    Certificate
)
```

Name	Type	Valid range	Description
<i>Certificate</i>	1609.2 certificate	Any 1609.2 certificate	The certificate to be added as a trust anchor

##### **9.4.2.1.3 When generated**

The primitive is generated as needed by a process to add a trust anchor to the SSME.

##### **9.4.2.1.4 Effect of receipt**

On receipt, the SSME determines whether *Certificate* has the following properties:

- It is a correctly formed certificate.
- It has not been revoked.
- If the IssuerIdentifier is of type `self`, the certificate verifies with the public key indicated by the `verifyKeyIndicator` field in theToBeSignedCertificate.

If all of these conditions hold, the SSME adds *Certificate* to its store of trust anchors. The SSME then invokes the corresponding confirm primitive to return the result of the processing.

If *Certificate* is an implicit certificate, subsequent processing might be made more efficient if the associated public key is also calculated and stored.

### **9.4.2.2 SSME-AddTrustAnchor.confirm**

#### **9.4.2.2.1 Function**

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### **9.4.2.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddTrustAnchor.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
Result Code	Enumerated	Success Invalid input Certificate revoked Certificate did not verify	Indicates the result of the associated request

#### **9.4.2.2.3 When generated**

The primitive is generated in response to the corresponding request.

#### **9.4.2.2.4 Effect of receipt**

No behavior is specified.

### **9.4.3 SSME-AddCertificate**

#### **9.4.3.1 SSME-AddCertificate.request**

##### **9.4.3.1.1 Function**

This primitive allows a process to add a certificate to the SSME's store of certificates.

##### **9.4.3.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddCertificate.request (
    Certificate,
    Verified
)
```

Name	Type	Valid range	Description
<i>Certificate</i>	1609.2 certificate	Any 1609.2 certificate	The certificate to be added
<i>Verified</i>	Boolean		Whether or not the certificate has been cryptographically verified (see 9.3.11.1.2)

#### 9.4.3.1.3 When generated

The primitive is generated as needed by any process or entity that wishes to add a certificate to the SSME.

#### 9.4.3.1.4 Effect of receipt

On receipt, the SSME determines whether *Certificate* is a correctly formed certificate. If it is:

- a) The SSME adds *Certificate* to its store of certificates.
- b) The SSME sets the *Last Received CRL Time* and *Next Expected CRL Time* to the values set by the most recent appropriate invocation of SSME-AddRevocationInfo.request.
- c) The SSME increments its count of the number of P2PCD responses with respect to  $c$ , the HashedId3 corresponding to *Certificate*, for each SDEE for which such a count is being maintained. In the terminology of D.4.2.1, the SDEE determines whether  $p2pcdResponseCount(c, s)$  exists for any SDEE ID  $s$ , and, if so, the SSME increments all instances of  $p2pcdResponseCount(c, s)$ .
- d) The SSME determines whether *Certificate* corresponds to any certificate which has been noted as a potential subject of a P2PCD learning request. If this is the case, it removes that certificate as a potential subject. In the terminology of D.4.2.1, the SSME determines whether there is an entry equal to  $h$  in *queuedMissingCertIndicators(s)* for any SDEE ID  $s$ , where  $h$  is the HashedId8 corresponding to *Certificate* and *queuedMissingCertIndicators* is the array specified in D.4.2.1.1. If so, the SSME removes  $h$  from all instances of *queuedMissingCertIndicators(s)* where it appears.

The SSME then invokes the corresponding confirm primitive to return the result of the processing.

#### 9.4.3.1.5 Effect of receipt

No behavior is specified.

### 9.4.3.2 SSME-AddCertificate.confirm

#### 9.4.3.2.1 Function

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### **9.4.3.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddCertificate.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Invalid input	Indicates the result of the associated request

#### **9.4.3.2.3 When generated**

The primitive is generated in response to the corresponding request.

#### **9.4.3.2.4 Effect of receipt**

No behavior is specified.

### **9.4.4 SSME-VerifyCertificate**

#### **9.4.4.1 SSME-VerifyCertificate.request**

##### **9.4.4.1.1 Function**

This primitive is used by a process to request the SSME to verify a certificate.

##### **9.4.4.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-VerifyCertificate.request (
    Certificate,
    Signed SPDU (optional)
)
```

Name	Type	Valid range	Description
<i>Certificate</i>	Certificate	A valid Certificate	The certificate to be verified
<i>Signed SPDU</i>	Ieee1609Dot2-Data	An Ieee1609Dot2Data of type signedData	If <i>Certificate</i> is an ImplicitCertificate, a signed SPDU that indicates it was signed with <i>Certificate</i>

##### **9.4.4.1.3 When generated**

The primitive is generated as needed within the WAVE Security Services or by other entities or processes.

#### **9.4.4.1.4 Effect of receipt**

On receipt, the SSME attempts to verify *Certificate* (if it is an ExplicitCertificate) or attempts to verify the signature on *signed SPDU* (if *Certificate* is an ImplicitCertificate). The result is returned via SSME-Verify-Certificate.confirm. If the certificate verifies correctly, the SSME updates the stored information about the certificate to indicate that it has been verified. If *Certificate* is an ExplicitCertificate and does not verify correctly, the SSME updates the stored information about the certificate to indicate that it has failed cryptographic verification. If *Certificate* is an ImplicitCertificate and does not verify correctly, the SSME takes no action because in this case it is ambiguous whether Certificate or signed SPDU is invalid.

#### **9.4.4.2 SSME-VerifyCertificate.confirm**

##### **9.4.4.2.1 Function**

This primitive returns the result of the corresponding request primitive.

##### **9.4.4.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-VerifyCertificate.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Verified No trust anchor Chain too long for implementation Not cryptographically valid Inconsistent permissions in chain Revoked Dubious Unsupported critical information fields Invalid encoding	The result of the verification request

##### **9.4.4.2.3 When generated**

The primitive is generated in response to SSME-VerifyCertificate.request. The parameters are set as follows. In the description below, “the input *XXX*” is shorthand for “the parameter *XXX* provided to the corresponding invocation of SSME-CertificateInfo.request”.

- a) If one of the following error conditions holds, *Result Code* is set to indicate that error condition as follows:
  - 1) “No trust anchor” if the chain from the certificate does not end at a known trust anchor (see 5.1.2.1).
  - 2) “Chain too long for implementation” if the chain is longer than the implementation supports (see 5.1.2.3).

- 3) “Not cryptographically valid” if any certificate in the chain fails to verify cryptographically (see 5.1.2.3).
  - 4) “Inconsistent permissions in chain” if the permissions in the chain are inconsistent (see 5.1.2.4).
  - 5) “Revoked” if the certificate has been revoked (see 5.1.3).
  - 6) “Dubious” if the revocation information relevant to the certificate is overdue (see 5.1.3.6).
  - 7) “Unsupported critical information fields” if the certificate or a certificate in its chain contains an unsupported critical information field (see 5.2.5).
  - 8) “Invalid encoding” if the certificate or a certificate in its chain is not a valid encoding of the data structures in Clause 6.
- b) If more than one of the error conditions in step a) holds, *Result Code* is set to indicate any one of the applicable error conditions.
  - c) Otherwise, *Result Code* is set to “verified and trusted”.

#### **9.4.4.2.4 Effect of receipt**

None specified.

### **9.4.5 SSME-DeleteCertificate**

#### **9.4.5.1 SSME-DeleteCertificate.request**

##### **9.4.5.1.1 Function**

This primitive allows a process to delete a certificate from the SSME’s store of certificates.

##### **9.4.5.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-DeleteCertificate.request (
    Identifier Type,
    Identifier
)
```

<b>Name</b>	<b>Type</b>	<b>Valid range</b>	<b>Description</b>
<i>Identifier Type</i>	Enumerated	Certificate HashedId8 HashedId10	Indicates the type of input data used to identify the certificate
<i>Identifier</i>	Octet string	Any	The encoded certificate, HashedId8, or HashedId10 identifying the certificate in question

##### **9.4.5.1.3 When generated**

The primitive is generated as needed by any trusted entity that communicates with the SSME.

#### **9.4.5.1.4 Effect of receipt**

On receipt, if *Identifier* identifies a certificate known to the SSME, the SSME deletes that certificate and all information associated with it.

#### **9.4.5.2 SSME-DeleteCertificate.confirm**

##### **9.4.5.2.1 Function**

The primitive returns the result of the corresponding request primitive.

##### **9.4.5.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-DeleteCertificate.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Invalid input	Indicates the result of the associated request

##### **9.4.5.2.3 When generated**

The primitive is generated in response to the corresponding request.

##### **9.4.5.2.4 Effect of receipt**

No behavior is specified.

#### **9.4.6 SSME-AddHashIdBasedRevocation**

##### **9.4.6.1 SSME-AddHashIdBasedRevocation.request**

##### **9.4.6.1.1 Function**

The primitive is used to provide the SSME with Hash ID-based revocation information relating to a certificate as specified in 5.1.3.5.

#### **9.4.6.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddHashIdBasedRevocation.request (
    Identifiers,
    CracaId,
    CRL Series,
    Expiry
)
```

Name	Type	Valid range	Description
<i>Identifiers</i>	Array of HashedId10	As stated under Type	The HashedId10 values identifying the revoked certificates
<i>CracaId</i>	HashedId8	An octet string of length 8	An identifier for the CRACA (see 5.1.3)
<i>CRL series</i>	Integer	$1\dots 2^{32}-1$	The CRL series that includes the revocation information
<i>Expiry</i>	Time	Any time in the future	The time at which the indicated revocation information may be removed

#### **9.4.6.1.3 When generated**

The primitive is generated by a process or entity that obtains certificate revocation information. This information may be obtained from a CRL or by other means out of the scope of this standard.

#### **9.4.6.1.4 Effect of receipt**

The SSME stores the revocation information and returns a confirm primitive.

If the revoked certificate is a CA certificate, the SSME may choose to locate any certificates issued by that CA within the SSME internal storage, and mark those certificates as also revoked. This may save time when processing subsequently received signed data, as it enables the SSME to identify a signing certificate as revoked immediately, rather than having to use the full processing given in this standard.

### **9.4.6.2 SSME-AddHashIdBasedRevocation.confirm**

#### **9.4.6.2.1 Function**

This primitive returns the result of the corresponding request.

#### **9.4.6.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddHashIdBasedRevocation.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Invalid input	Indicates the result of the associated request

#### **9.4.6.2.3 When generated**

The primitive is generated in response to the corresponding request.

#### **9.4.6.2.4 Effect of receipt**

No behavior is specified.

### **9.4.7 SSME-AddIndividualLinkageBasedRevocation**

#### **9.4.7.1 SSME-AddIndividualLinkageBasedRevocation.request**

##### **9.4.7.1.1 Function**

The primitive is used by a process to provide the SSME with individual linkage based revocation information as specified in 5.1.3.4.

##### **9.4.7.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddIndividualLinkageBasedRevocation.request (
    CracalId,
    CRL Series,
    RevocationInfos
)
```

<b>Name</b>	<b>Type</b>	<b>Valid range</b>	<b>Description</b>
<i>CracaId</i>	HashedId8	An octet string of length 8	An identifier for the CRACA (see 5.1.3)
<i>CRL series</i>	Integer	$1 \dots 2^{32} - 1$	The CRL series that includes the revocation information
<i>RevocationInfos</i>	Array of RevocationInfo, each containing the entries below		
	<i>iRev</i>	Integer	$0..2^{16} - 1$ An indication of the time period when the revocation information becomes effective
	<i>iMax</i>	Integer	$0..2^{16} - 1$ An indication of the time period when the revocation information stops being effective
	<i>jMax</i>	Integer	$0..2^8 - 1$ The number of certificates within each time period
	<i>Linkage Seed 1</i>	Octet String	Octet string of length 16 The linkage seed from the first linkage authority
	<i>Linkage Authority Identifier 1</i>	Octet String	Octet string of length 2 An indication of the linkage authority that generated linkage seed 1, an octet string of length 2
	<i>Linkage Seed 2</i>	Octet String	Octet string of length 16 The linkage seed from the second linkage authority
	<i>Linkage Authority Identifier 2</i>	Octet String	Octet string of length 2 An indication of the linkage authority that generated linkage seed 2, an octet string of length 2

#### 9.4.7.1.3 When generated

The primitive is generated by a process or entity that obtains certificate revocation information. This information may be obtained from a CRL or by other means out of the scope of this standard.

#### 9.4.7.1.4 Effect of receipt

The SSME stores the revocation information and returns a confirm primitive.

### 9.4.7.2 SSME-AddIndividualLinkageBasedRevocation.confirm

#### 9.4.7.2.1 Function

This primitive returns the result of the corresponding request.

#### 9.4.7.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-AddIndividualLinkageBasedRevocation.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Invalid input	Indicates the result of the associated request

#### **9.4.7.2.3 When generated**

The primitive is generated in response to the corresponding request.

#### **9.4.7.2.4 Effect of receipt**

No behavior is specified.

### **9.4.8 SSME-AddGroupLinkageBasedRevocation**

#### **9.4.8.1 SSME-AddGroupLinkageBasedRevocation.request**

##### **9.4.8.1.1 Function**

The primitive is used by a process to provide the SSME with group linkage based revocation information as specified in 5.1.3.4.

##### **9.4.8.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddGroupLinkageBasedRevocation.request (
    CracaId,
    CRL Series,
    RevocationInfos
)
```

Name	Type	Valid range	Description
<i>Cracald</i>	HashedId8	An octet string of length 8	An identifier for the CRACA (see 5.1.3)
<i>CRL series</i>	Integer	$1 \dots 2^{32} - 1$	The CRL series that includes the revocation information
<i>RevocationInfos</i>	Array of RevocationInfo, each containing the entries below		
<i>iRev</i>	Integer	$0..2^{16} - 1$	An indication of the time period when the revocation information becomes effective
<i>iMax</i>	Integer	$0..2^{16} - 1$	An indication of the time period when the revocation information stops being effective
<i>Linkage Seed 1</i>	Octet string	Octet string of length 16	The linkage seed from the first linkage authority
<i>Linkage Authority Identifier 1</i>	Octet string	Octet string of length 2	An indication of the linkage authority that generated linkage seed 1, an octet string of length 2
<i>Linkage Seed 2</i>	Octet string	Octet string of length 16	The linkage seed from the second linkage authority
<i>Linkage Authority Identifier 2</i>	Octet string	Octet string of length 2	An indication of the linkage authority that generated linkage seed 2, an octet string of length 2

#### 9.4.8.1.3 When generated

The primitive is generated by a process or entity that obtains certificate revocation information. This information may be obtained from a CRL or by other means out of the scope of this standard.

#### 9.4.8.1.4 Effect of receipt

The SSME stores the revocation information and returns a confirm primitive.

### 9.4.8.2 SSME-AddGroupLinkageBasedRevocation.confirm

#### 9.4.8.2.1 Function

This primitive returns the result of the corresponding request.

#### 9.4.8.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-AddGroupLinkageBasedRevocation.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Invalid input	Indicates the result of the associated request

#### **9.4.8.2.3 When generated**

The primitive is generated in response to the corresponding request.

#### **9.4.8.2.4 Effect of receipt**

No behavior is specified.

### **9.4.9 SSME-AddRevocationInfo**

#### **9.4.9.1 SSME-AddRevocationInfo.request**

##### **9.4.9.1.1 Function**

The primitive is used to update the SSME's information about the status of a series of revocation information.

##### **9.4.9.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-AddRevocationInfo.request (
    CRL Type,
    CRL Series,
    CRACA ID,
    Issue Date,
    Next Crl
)
```

Name	Type	Valid range	Description
<i>CRL Type</i>	Enumerated value	Id Only Id Expiry	The type of the revocation information
<i>CRL Series</i>	Integer	$1\dots 2^{32}-1$	The CRL series value from the revocation information
<i>CRACA ID</i>	Octet string	An octet string of length 8	The low-order eight octets of the hash of the CRACA ID associated with the revocation information
<i>Issue Date</i>	Time	Any date prior to current date	The issue date of the revocation information
<i>Next CRL</i>	Time	Any time after <i>Issue Date</i>	The time when the next revocation information in this series is expected to be issued

#### **9.4.9.1.3 When generated**

This primitive is generated by an entity that receives revocation information.

#### **9.4.9.1.4 Effect of receipt**

On receipt of this primitive the SSME updates the CRL information as listed above. For every known certificate that might be present on the CRL, the SSME updates the *Last Received CRL Time* to *Issue Date* and the *Next Expected CRL Time* to *Next CRL*.

### **9.4.9.2 SSME-AddRevocationInfo.confirm**

#### **9.4.9.2.1 Function**

The primitive acknowledges the receipt of the corresponding request primitive.

#### **9.4.9.2.2 Semantics of the service primitive**

This primitive takes no parameters.

#### **9.4.9.2.3 When generated**

This primitive is generated in response to the corresponding request primitive.

#### **9.4.9.2.4 Effect of receipt**

No behavior is specified.

## 9.4.10 SSME-RevocationInformationStatus

### 9.4.10.1 SSME-RevocationInformationStatus.request

#### 9.4.10.1.1 Function

The primitive is used by a process to request CRL information.

#### 9.4.10.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-RevocationInformationStatus.request (
    CRL Series,
    CRACA ID
)
```

Name	Type	Valid range	Description
<i>CRL Series</i>	Integer	$1 \dots 2^{32} - 1$	The CRL series value from the revocation information
<i>CRACA ID</i>	Octet string	An octet string of length 8	The low-order eight octets of the hash of the CRACA certificate

#### 9.4.10.1.3 When generated

This primitive is generated by any entity to request the status of revocation information.

#### 9.4.10.1.4 Effect of receipt

On receipt of this primitive, the SSME retrieves information about the revocation information identified by *CRL Series* and *CRACA ID*. It returns that information, if available, to the process via SSME-RevocationInformationStatus.confirm.

### 9.4.10.2 SSME-RevocationInformationStatus.confirm

#### 9.4.10.2.1 Function

The primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### 9.4.10.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-RevocationInformationStatus.confirm (
    Result Code,
    Revocation Type,
    Issue Date,
    Next CRL
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success Unrecognized identifier Expired Not issued yet Missing	Indicates the result of the associated request
<i>Revocation Type</i>	Enumerated value	Hash ID based Linkage ID based	The type of the entries in the CRL
<i>Issue Date</i>	Time	Any date prior to current date	The issue date of the CRL
<i>Next CRL</i>	Time	Any time after <i>Issue Date</i>	The time when the next CRL is expected to be issued

#### 9.4.10.2.3 When generated

This primitive is generated in response to the corresponding request primitive.

#### 9.4.10.2.4 Effect of receipt

No behavior is specified.

### 9.4.11 SSME-P2PcdResponseGenerationService

#### 9.4.11.1 SSME-P2pcdResponseGenerationService.request

##### 9.4.11.1.1 Function

This primitive indicates that a P2PCD Entity requests to be notified when a P2PCD learning response is to be generated.

##### 9.4.11.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-P2pcdResponseGenerationService.request (
    SDEEs
)
```

Name	Type	Valid range	Description
<i>SDEEs</i>	Array of integer, or “all”	Any	The identifier of the SDEEs for which the invoking P2PCD process wishes to receive response generation indications

##### 9.4.11.1.3 When generated

The primitive is generated as needed by a P2PCD process.

#### **9.4.11.1.4 Effect of receipt**

On receipt, the SSME generates a SSME-P2pcdResponseGenerationService.confirm indicating whether the request is accepted. On acceptance, the invoking P2PCD process receives response generation indications via SSME-P2pcdResponseGeneration.indication.

#### **9.4.11.2 SSME-P2pcdResponseGenerationService.confirm**

##### **9.4.11.2.1 Function**

This primitive confirms the acceptance of the corresponding request.

##### **9.4.11.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-P2pcdResponseGenerationService.confirm (
    SDEE IDs
)
```

Name	Type	Valid range	Description
SDEE IDs	Array of integers, or “all”		Indicates the SDEEs for which the P2PCD process receives response generation indications

##### **9.4.11.2.3 When generated**

The primitive is generated in response to SSME-P2pcdResponseGenerationService.request.

##### **9.4.11.2.4 Effect of receipt**

The P2PCD Entity may take action based on the confirmation.

#### **9.4.12 SSME-P2pcdResponseGeneration**

##### **9.4.12.1 SSME-P2pcdResponseGeneration.indication**

##### **9.4.12.1.1 Function**

This primitive indicates that the conditions for generation of a P2PCD response have been met.

#### **9.4.12.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-P2pcdResponseGeneration.indication (
    Certificates
)
```

Name	Type	Valid range	Description
<i>Certificates</i>	Array of validly encoded Certificate		The certificates to be included in the P2PCD response

#### **9.4.12.1.3 When generated**

The primitive is generated when the conditions for generation of a P2PCD response have been met.

#### **9.4.12.1.4 Effect of receipt**

The receiving P2PCD Entity may generate and send an IEEE1609dot2Peer2PeerPDU as defined in 8.4.1.

### **9.4.13 SSME-P2pcdConfiguration**

#### **9.4.13.1 SSME-P2pcdConfiguration.request**

##### **9.4.13.1.1 Function**

This primitive allows an invoking entity to update the peer-to-peer certificate distribution parameters relevant to a particular SDEE as specified in 8.2.3.

##### **9.4.13.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-P2pcdConfiguration.request (
    SDEE ID,
    p2pcd_maxResponseBackoff,
    p2pcd_responseActiveTimeout,
    p2pcd_requestActiveTimeout,
    p2pcd_observedRequestTimeout,
    p2pcd_currentlyUsedTriggerCertificateTime,
    p2pcd_responseCountThreshold
)
```

Name	Type	Valid range	Description
<i>SDEE ID</i>	Integer or “all”	Any	The ID of the SDEE for which this configuration applies, or “all”
<i>p2pcd_max-ResponseBackoff</i>	Time	Any positive time	The maximum backoff time when responding to a request
<i>p2pcd_response-ActiveTimeout</i>	Time	Any positive time	The time after which a response-active state ends with respect to a particular trigger certificate
<i>p2pcd_request-ActiveTimeout</i>	Time	Any positive time	The time after which a request-active state ends with respect to a particular trigger certificate
<i>p2pcd_observed-RequestTimeout</i>	Time	Any positive time	The time after which a request-active state ends with respect to certificate indicated in a P2PCD learning request from a different WAVE device
<i>p2pcd_currentlyUsed-TriggerCertificateTime</i>	Time	Any positive time	The time used to determine whether a trigger certificate is “currently used”.
<i>p2pcd_response-CountThreshold</i>	Integer	Any integer > 0	A number used to determine whether or not a response is sent to a particular P2PCD request

#### 9.4.13.1.3 When generated

The primitive is generated as needed by a management process.

#### 9.4.13.1.4 Effect of receipt

When received the SSME updates the indicated parameters with the indicated values.

### 9.4.13.2 SSME-P2pcdConfiguration.confirm

#### 9.4.13.2.1 Function

This primitive returns the values calculated in the processing specified for the corresponding request primitive.

#### 9.4.13.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-P2pcdConfiguration.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Success, failure	Indicates the result of the associated request

#### **9.4.13.2.3 When generated**

The primitive is generated in response to SSME-P2pcdConfiguration.request.

#### **9.4.13.2.4 Effect of receipt**

None specified.

### **9.5 SSME-Sec SAP**

#### **9.5.1 SSME-Sec-ReplayDetection**

##### **9.5.1.1 SSME-Sec-ReplayDetection.request**

###### **9.5.1.1.1 Function**

This primitive allows any SDEE to determine whether received signed data is a duplicate of signed data that has already been received by that entity, and to request the SSME to store that signed data for future replay detection.

###### **9.5.1.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-Sec-ReplayDetection.request (
    SDEE ID,
    Data,
    Discard Time
)
```

Name	Type	Valid range	Description
<i>SDEE ID</i>	Integer	Any	The SDEE ID that identifies the SDEE
<i>Data</i>	Octet string	An octet string	The encoded SignedData that is to be checked for being a replay
<i>Discard Time</i>	Time	Any time in the future	The time at which the data provided as the <i>Data</i> parameter may be discarded and no longer checked for discard

###### **9.5.1.1.3 When generated**

The primitive is generated during the execution of Sec-SignedDataVerification.request as specified in the specification of that primitive.

#### **9.5.1.1.4 Effect of receipt**

On receipt of this primitive, the SSME determines whether the input *Data* has already been received by the entity indicated by the *SDEE ID*, and stores *Data* for use in future replay detection. The result of the replay detection is returned to the SDS by use of the corresponding confirm primitive.

The following processing gives correct output from the corresponding confirm primitive.

- a) Create the variable *Result Code*, to be used to return the result of the replay detection operation to the SDS.
- b) If a set (*SDEE ID*, *Data*, *Discard Time*) have already been stored by the SSME, and if the current time is not later than *Discard Time*, set *Result Code* to “replay”. Otherwise, set *Result Code* to “not replay”.
- c) Store the set (*SDEE ID*, *Data*, *Discard Time*).
- d) Invoke the corresponding confirm primitive to return *Result Code*.

NOTE—An implementation has the option of no longer storing (*SDEE ID*, *Data*, *Discard Time*) after *Discard Time* has passed.

#### **9.5.1.2 SSME-Sec-ReplayDetection.confirm**

##### **9.5.1.2.1 Function**

This primitive returns the values calculated in the processing specified for the corresponding request primitive.

##### **9.5.1.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-Sec-ReplayDetection.confirm (
    Result Code
)
```

Name	Type	Valid range	Description
<i>Result Code</i>	Enumerated	Replay Not replay	Indicates the result of the associated request

##### **9.5.1.2.3 When generated**

The primitive is generated in response to SSME-Sec-ReplayDetection.request.

##### **9.5.1.2.4 Effect of receipt**

Specified in the specification of the invoking process.

## **9.5.2 SSME-Sec-IncomingP2pcdInfo**

### **9.5.2.1 SSME-Sec-IncomingP2pcdInfo.request**

#### **9.5.2.1.1 Function**

This primitive is used by the SSME to determine whether to initiate activities associated with peer-to-peer certificate distribution as a result of a signed SPDU received by a SDEE and passed to Sec-SecureData-Preprocessing.request.

#### **9.5.2.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-Sec-IncomingP2pcdInfo.request (
    SDEE ID,
    Certificate (optional),
    P2pcdLearningRequest (optional)
)
```

Name	Type	Valid range	Description
SDEE ID	Integer	Any	The ID of the SDEE ID that invoked Sec-SecureDataPreprocessing.request
Certificate	Certificate	An array of encoded Certificate as defined in 6.4.2	The certificate field from the SignerIdentifier that signed the signed SPDU
P2pcdLearningRequest	HashedId3	Any	The p2pcdLearningRequest field from the signed SPDU

#### **9.5.2.1.3 When generated**

The primitive is generated by the SDS if necessary in the course of executing Sec-SecureData-Preprocessing.request.

#### **9.5.2.1.4 Effect of receipt**

On receipt of this primitive, the SSME carries out the operations specified in 8.2.4.1, step a)2), and 8.2.4.2, step a)1). One possible implementation of this is specified in D.4.3.2.

### **9.5.2.2 SSME-Sec-IncomingP2pcdInfo.confirm**

#### **9.5.2.2.1 Function**

This primitive confirms the reception of the corresponding request primitive and informs the caller of state changes made as a result.

### **9.5.2.2.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-Sec-IncomingP2pcdInfo.confirm (
    Request Active for Certificate
    Request Active for P2PCD Learning Request
    Response Active for P2PCD Learning Request
)
```

Name	Type	Valid range	Description
<i>Request Active for Certificate</i>	Boolean	True, False	Whether the SSME is in a request-active state for the certificate in the signed SPDU
<i>Request Active for P2PCD Learning Request</i>	Boolean	True, False	Whether the SSME is in a request-active state for the p2pcdLearningRequest in the signed SPDU
<i>Response Active for P2PCD Learning Request</i>	Boolean	True, False	Whether the SSME is in a response-active state for the p2pcdLearningRequest in the signed SPDU

### **9.5.2.2.3 When generated**

The primitive is generated in response to SSME-Sec-IncomingP2pcdInfo.request.

### **9.5.2.2.4 Effect of receipt**

Specified in the specification of the invoking process.

## **9.5.3 SSME-Sec-OutgoingP2pcdInfo**

### **9.5.3.1 SSME-Sec-OutgoingP2pcdInfo.request**

#### **9.5.3.1.1 Function**

This primitive is used by the SDS to request the SSME to provide a p2pcdLearningRequest for inclusion in a signed SPDU.

#### **9.5.3.1.2 Semantics of the service primitive**

The parameters of the primitive are as follows:

```
SSME-Sec-OutgoingP2pcdInfo.request (
    SDEE ID,
    Certificate
)
```

Name	Type	Valid range	Description
SDEE ID	Integer	Any	The ID of the SDEE ID that invoked Sec-SecureDataPreprocessing.request
Certificate	Certificate	An encoded Certificate as defined in 6.4.2	The certificate that is to be used to sign the signed SPDU

#### 9.5.3.1.3 When generated

The primitive is generated by the SDS in the course of executing Sec-SignedData.request.

#### 9.5.3.1.4 Effect of receipt

On receipt of this primitive, the SSME carries out the operations specified in 8.2.4.1, step b)1). One possible implementation of this is specified in D.4.3.3.

#### 9.5.3.2 SSME-Sec-OutgoingP2pcdInfo.confirm

##### 9.5.3.2.1 Function

This primitive returns the values calculated in the processing specified for the corresponding request primitive.

##### 9.5.3.2.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
SSME-Sec-OutgoingP2pcdInfo.confirm (
    p2pcdLearningRequest (optional)
)
```

Name	Type	Valid range	Description
p2pcdLearningRequest	HashedId3		The p2pcdLearningRequest value to be included in the signed SPDU. If this is omitted, no p2pcdLearningRequest value is to be included.

##### 9.5.3.2.3 When generated

The primitive is generated in response to SSME-Sec-OutgoingP2pcdInfo.request.

##### 9.5.3.2.4 Effect of receipt

The *P2PCD Learning Request* is included in the signed SPDU that is being generated.

## Annex A

(normative)

### Protocol Implementation Conformance Statement (PICS) proforma

#### A.1 Instructions for completing the PICS proforma

##### A.1.1 General structure of the PICS proforma

The first parts of the PICS proforma, Implementation identification and Protocol summary, are to be completed as indicated with the information necessary to identify fully both the supplier and the implementation.

The main part of the PICS proforma is a fixed questionnaire, divided into subclauses, each containing a number of individual items. Answers to the questionnaire items are to be provided in the rightmost column, either by simply marking an answer to indicate a restricted choice (usually Yes or No) or by entering a value or a set or a range of values. If there are items where two or more choices from a set of possible answers may apply, all relevant choices are to be marked.

Each item is identified by an item reference in the first column. The second column contains the question to be answered. The third column contains the reference or references to the material that specifies the item in the main body of this standard. The remaining columns record the status of each item, i.e., whether support is mandatory, optional, or conditional, and provide the space for the answers. Marking an item as supported is to be interpreted as a statement that all relevant requirements of the subclauses and normative annexes, cited in the References column for the item, are met by the implementation.

A supplier may also provide, or be required to provide, further information, categorized as either Additional Information or Exception Information. When present, each kind of further information is to be provided in a further subclause of items labeled A<I> or X<I>, respectively, for cross-referencing purposes, where <I> is any unambiguous identification for the item (e.g., simply a numeral). There are no other restrictions on its format or presentation.

A completed PICS proforma, including any Additional Information and Exception Information, is the PICS for the implementation in question.

NOTE—Where an implementation is capable of being configured in more than one way, a single PICS may be able to describe all such configurations. However, the supplier has the choice of providing more than one PICS, each covering some subset of the implementation's capabilities, if this makes for easier and clearer presentation of the information.

##### A.1.2 Additional information

Items of Additional Information allow a supplier to provide further information intended to assist in the interpretation of the PICS. It is not intended or expected that a large quantity of information will be supplied, and a PICS can be considered complete without any such information. Examples of such Additional Information might be an outline of the ways in which an (single) implementation can be set up to operate in a variety of environments and configurations, or information about aspects of the implementation that are outside the scope of this standard but have a bearing upon the answers to some items.

References to items of Additional Information may be entered next to any answer in the questionnaire, and may be included in items of Exception Information.

### A.1.3 Exception information

It may happen occasionally that a supplier will wish to answer an item with mandatory status (after any conditions have been applied) in a way that conflicts with the indicated requirement. No preprinted answer will be found in the Support column for this. Instead, the supplier shall write the missing answer into the Support column, together with an X<I> reference to an item of Exception Information, and shall provide the appropriate rationale in the Exception Information item itself.

An implementation for which an Exception Information item is required in this way does not conform to this standard.

NOTE—A possible reason for the situation described above is that a defect in this standard has been reported, a correction for which is expected to change the requirement not met by the implementation.

### A.1.4 Conditional status

The PICS proforma contains a number of conditional items. These are items for which both the applicability of the item itself, and its status if it does apply, mandatory or optional, are dependent upon whether or not certain other items are supported.

A conditional symbol is of the form “<pred>:<S>”, where “<pred>” is a predicate as specified below, and “<S>” is one of the status symbols C, M, or O.

If the value of the predicate is true, the conditional item is applicable, and its status is given by S, then the support column is to be completed in the usual way. Otherwise, the conditional item is not relevant.

A predicate is one of the following:

- An item-reference for an item in the PICS proforma: the value of the predicate is true if the item is marked as supported, and is false otherwise.
- A Boolean expression constructed by combining item-references using the boolean operator OR: The value of the predicate is true if one or more of the items is marked as supported, and is false otherwise. For compactness, item-references combined with a comma are considered to be combined with the OR operator.
- An item-reference or combination of item references as described in the previous two dashed items, followed by “<rel> <num>”, such that:
  - The relationship “<rel>” is “<”, “=”, or “>”, indicating “less than”, “equal to”, or “greater than” <num>.
  - The number “<num>” is an integer.
  - The predicate is true if the item-reference is true as defined above and the value defined in the item body matches the numeric relationship indicated by “<rel> <num>”, and the predicate is false if either the item-reference is not true as defined above or the value defined in the item body does not match the numeric relationship indicated by “<rel> <num>”. Example: For the item S1.2.2.5.1.2.1, “Maximum number of rectangularRegions supported”, the Status “S1.2.2.5.1.2:8:M > 8:O” indicates that if item S1.2.2.5.1.2, “Support a rectangular region”, is supported, then item S1.2.2.5.1.2.1, “Maximum number of rectangularRegions supported”, shall have a value of at least 8 and may have a value greater than 8.

A status of C<n> indicates a mutual conditionality such that support of one and one only of the items that have the same predicate and status C<n> is mandatory.

A status of O<n> indicates a mutual conditionality such that the feature is optional but that support of at least one of the items that have the same predicate and status O<n> is mandatory.

A status of M indicates that the feature is mandatory.

A status of O indicates that the feature is optional.

## **A.2 PICS proforma—IEEE Std 1609.2<sup>12</sup>**

### **A.2.1 Identification**

Only the first three items are required for all implementations. Other information may be completed as appropriate in meeting the requirement for full identification.

The terms *name* and *version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., type, series, model).

Supplier	
Contact point for queries about the PICS	
Implementation name(s) and version(s)	
Other information necessary for full identification, e.g., name(s) and version(s) of the machines and/or operating systems(s), system names	

---

<sup>12</sup> Copyright release for PICS proforma: Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

## A.2.2 Protocol summary

Identification of protocol standard	IEEE Std 1609.2	
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	Amd. :	Corr. :
Have any exception items been required? (See A.1.3; the answer <i>Yes</i> means that the implementation does not conform to IEEE Std 1609.2)	<input type="checkbox"/> Yes	<input type="checkbox"/> No
Date of statement (dd/mm/yy)		

## A.2.3 Conformance statement

### A.2.3.1 Security services

This presents a list of the security functionality that an implementation may claim to support.

Item	Security configuration (top-level)	Reference	Status	Support
S1.	Support secure data service		O1	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.1.	<b>Secure data exchange entity (SDEE) identification</b>	4.2.2.1	S1:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.1.1.	Support only one SDEE	4.2.2.1	S1.1:C1	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.1.2.	Distinguish between SDEEs	4.2.2.1	S1.1:C1	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.	<b>Generate secured protocol data unit (SPDU)</b>		S1:O2	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.1.	Create Ieee1609Dot2Data containing unsecured data	4.2.2.2.2	S1.2:O3	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.	Create Ieee1609Dot2Data containing valid SignedData	4.2.2.2.3, 5.2, 5.3.1, 5.3.3, 5.3.7, 6.3.4, 6.3.9, 9.3.9.1	S1.2:O3	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.1.	Using a valid HashAlgorithm	6.3.5	S1.2.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.1.1.	Support signing with hash algorithm SHA-256	6.3.5	S1.2.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.1.2.	Support signing with hash algorithm other than SHA-256	6.3.5	S1.2.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.2.	Containing a Signed Data payload	6.3.6	S1.2.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.2.1.	... with payload containing data	6.3.7	S1.2.2.2:O4	<input type="checkbox"/> Yes <input type="checkbox"/> No

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Support</b>
S1.2.2.2.	... with payload containing extDataHash	6.3.7	S1.2.2.2: O4	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.3.	... with generationTime in the security headers	6.3.9, 6.3.11	S1.2.2.2: O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.	... with expiryTime in the security headers	6.3.9, 6.3.11	S1.2.2.2: O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.	... with generationLocation in the security headers	6.3.9, 6.3.12	S1.2.2.2: O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.6.	... with p2pcdLearningRequest in the security headers	6.3.9, 6.3.25	S1.2.2.2: O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.7.	... with missingCrlIdentifier in the security headers	6.3.9, 6.3.16	S1.2.2.2: O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.8.	... with encryptionKey in the security headers	6.3.9, 6.3.18	S1.2.2.2: O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.8.1.	.... ... with a PublicEncryptionKey	6.3.9, 6.3.18, 6.3.19	S1.2.2.2.8:O5	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.8.2.	.... ... with a SymmetricEncryptionKey	6.3.9, 6.3.18, 6.3.20	S1.2.2.2.8:O5	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.3.	Support a SignerIdentifier	6.3.24	S1.2.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.3.1.	... of type digest	6.3.26	S1.2.2.3:O6	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.3.2.	... of type certificate	6.4.2	S1.2.2.3:O6	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.3.2.1.	.... ... maximum number of Certificates in the chain	5.1.2.2	S1.2.2.3.2 8:M >8:O	Enter number: ( )
S1.2.2.3.3.	... of type self	6.3.24	S1.2.2.3:O6	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.	Support a Signature	6.3.28	S1.2.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.1.	... a ecdsa256Signature	6.3.29	S1.2.2.4:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.1.1.	.... ... using NIST p256	6.3.29	S1.2.2.4.1:O7	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.1.2.	.... ... using Brainpool p256r1	6.3.29	S1.2.2.4.1:O7	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.1.3.	.... ... with a x-only <i>r</i> value	6.3.23	S1.2.2.4.1:O8	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.1.4.	.... ... with a compressed <i>r</i> value	6.3.23	S1.2.2.4.1:O8	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.4.1.5.	.... ... with an uncompressed <i>r</i> value	6.3.23	S1.2.2.4.1:O8	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.	Determine that certificate used to sign data is valid (part of a consistent chain, valid at the current time and location, hasn't been revoked)	5.2, 6.4.2	S1.2.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.1.	Determine that the region is correct	6.4.8, 6.4.17	S1.2.2.5:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.1.1.	Support a circularRegion	6.4.17, 6.4.18	S1.2.2.5.1:O9	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.1.2.	Support a rectangularRegion	6.4.17, 6.4.20	S1.2.2.5.1:O9	<input type="checkbox"/> Yes <input type="checkbox"/> No

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Support</b>
S1.2.2.5.1.2.1.	Maximum number of rectangularRegions supported	6.4.17, 6.4.20	S1.2.2.5.1.2 8:M >8:O	Enter number: ( )
S1.2.2.5.1.3.	Support a polygonalRegion	6.4.17, 6.4.21	S1.2.2.5.1:O9	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.1.3.1.	Maximum number of points in a polygonalRegion	6.4.17, 6.4.21	S1.2.2.5.1.3 8:M >8:O	Enter number: ( )
S1.2.2.5.1.4.	Support identifiedRegion	6.4.17, 6.4.22	S1.2.2.5.1:O9	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.1.4.1.	Maximum number of identifiedRegions supported	6.4.17, 6.4.22	S1.2.2.5.1.4: 8:M >8:O	Enter number: ( )
S1.2.2.5.1.4.2.	Support IdentifiedRegion of type CountryOnly	6.4.22, 6.4.23	S1.2.2.5.1.4:O10	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.1.4.3.	Support IdentifiedRegion of type CountryAndRegions	6.4.22, 6.4.24	S1.2.2.5.1.4:O10	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.1.4.4.	Support IdentifiedRegion of type CountryAndSubregions	6.4.22, 6.4.25	S1.2.2.5.1.4:O10	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.2.	Determine that the certificate has the proper appPermissions	6.4.8, 6.4.28	S1.2.2.5:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.5.2.1.	Maximum number of PsidSsp in the appPermissions sequence	6.4.8, 6.4.28	S1.2.2.5.2 8:M >8:O	Enter number: ( )
S1.2.2.6.	Determine that key and certificate used to sign are a valid pair	5.3.7	S1.2.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.7.	Support signing with explicit certificates	6.4.6	S1.2.2.5:O11	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.8.	Support signing with implicit certificates	5.3.2, 6.4.5	S1.2.2.5:O11	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.2.9.	Generate elliptic curve digital signature algorithm (ECDSA) keypairs using a high-quality random number generator	5.3.6	S1.2.2.4.1: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.	Create Ieee1609Dot2Data containing EncryptedData	4.2.2.3.2, 5.3.4, 6.3.30	S1.2:O2	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.1.	Generate Elliptic Curve Integrated Encryption Scheme (ECIES) ephemeral keypairs using a high-quality random number generator	5.3.4, 5.3.5, 5.3.6	S1.3.3: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.2.	Maximum number of recipients supported	6.3.30	S1.2.3 8:M >8:O	Enter number: ( )
S1.2.3.3.	Containing PreSharedKeyRecipientInfo	6.3.31, 6.3.1	S1.2.3.2:O12	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.3.1.	Containing symmRecipientInfo	6.3.31, 6.3.2	S1.2.3.2:O12	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.3.2.	Containing certRecipientInfo	6.3.31, 6.3.3	S1.2.3.2:O12	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.3.3.	Containing signedDataRecipientInfo	6.3.31, 6.3.3	S1.2.3.2:O12	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.3.4.	Containing rekRecipientInfo	6.3.31, 6.3.3	S1.2.3.2:O12	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.	Support public-key encryption	6.3.5	S1.2.3:O13	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.1.	... using ECIES-256	6.3.5	S1.2.3.4:M	<input type="checkbox"/> Yes <input type="checkbox"/> No

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Support</b>
S1.2.3.4.1.1.	.... .... using NIST p256	6.3.5	S1.2.3.4.1:O14	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.1.2.	.... .... using Brainpool p256r1	6.3.5	S1.2.3.4.1:O14	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.1.3.	Support encrypting to an uncompressed encryption key	6.3.18	S1.2.3.4.1:O15	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.1.4.	Support encrypting to a compressed encryption key	6.3.18	S1.2.3.4.1:O15	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.1.5.	Support encrypting to an encryption key included in an explicit cert	6.3.18	S1.2.3.4.1:O16	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.1.6.	Support encrypting to an encryption key included in an implicit cert	6.3.18	S1.2.3.4.1:O16	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.4.2.	... using a different algorithm introduced at a later date	6.3.6	S1.2.3.4:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.5.	Support symmetric encryption	6.3.7	S1.2.3:O13	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.5.1.	... using AES-128	5.3.8, 6.3.7	S1.2.3.5:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.2.3.5.2.	... using a different algorithm introduced at a later date	6.3.3	S1.2.3.5:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.	<b>Receive secured protocol data unit (SPDU)</b>		S1:O2	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.1.	Support preprocessing SPDUs	4.2.2.3.1	S1.3.2.3.1, S3.2 S3.3:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.	Verify IEEE1609Dot2Data containing SignedData	4.2.2.3.2, 5.2, 5.3.1, 5.3.3, 5.3.7, 6.3.4, 6.3.9	S1.3:O17	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.1.	Using a valid HashAlgorithm		S1.3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.1.1.	Verify signed data using HashAlgorithm SHA-256	6.3.5	S1.3.2.1:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.1.2.	Verify signed data using a HashAlgorithm other than SHA-256	6.3.5	S1.3.2.1:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.	Containing a Signed Data payload	6.3.6	S1.3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.1.	... with payload containing data	6.3.7	S1.3.2.2:O18	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.2.	... with payload containing extDataHash	6.3.7	S1.3.2.2:O18	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.3.	... with generationTime in the security headers	6.3.9, 6.3.11	S1.3.2.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.4.	... with expiryTime in the security headers	6.3.9, 6.3.11	S1.3.2.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.5.	... with generationLocation in the security headers	6.3.9, 6.3.12	S1.3.2.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.6.	... with missingCertIdentifier in the security headers	6.3.9, 6.3.25	S1.3.2.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.7.	... with missingCrlIdentifier in the security headers	6.3.9, 6.3.16	S1.3.2.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.8.	... with encryptionKey in the security headers	6.3.9, 6.3.18	S1.3.2.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Support</b>
S1.3.2.2.8.1.	.... .... with a PublicEncryptionKey	6.3.9, 6.3.18, 6.3.19	S1.3.2.2.8:O19	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.2.8.2.	.... .... with a SymmetricEncryptionKey	6.3.9, 6.3.18, 6.3.20	S1.3.2.2.8:O19	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.3.	Support a SignerIdentifier	6.3.24	S1.3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.3.1.	... of type digest	6.3.26	S1.3.2.3:O20	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.3.2.	... of type certificate	6.4.2	S1.3.2.3:O20	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.3.2.1.	.... .... maximum number of Certificates in the chain	5.1.2.2	S1.3.2.3.2 1:M > 1:O	Enter number: ( )
S1.3.2.3.3.	... of type self		S1.3.2.3:O20	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.	Support a Signature	6.3.28	S1.3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.	... a ecdsa256Signature	6.3.29	S1.3.2.4:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.1.	.... .... using NIST p256	6.3.29	S1.3.2.4.1:O21	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.2.	.... .... using Brainpool p256r1	6.3.29	S1.3.2.4.1:O21	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.3.	.... .... with a x-only <i>r</i> value	6.3.23	S1.3.2.4.1:O22	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.4.	.... .... with a compressed <i>r</i> value	6.3.23	S1.3.2.4.1:O22	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.5.	.... .... with a compressed <i>r</i> value and fast verification	6.3.23	S1.3.2.4.1:O22	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.6.	.... .... with an uncompressed <i>r</i> value	6.3.23	S1.3.2.4.1:O22	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.4.1.7.	.... .... with an uncompressed <i>r</i> value and fast verification	6.3.23	S1.3.2.4.1:O22	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.	SignedData verification fails if the certificate is not valid (part of a consistent chain, valid at the current time and location, hasn't been revoked)	5.2, 6.4.2	S1.3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.	Reject data based on generation location being inconsistent with certificate	6.4.8, 6.4.17	S1.3.2.5:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.1.	... using a circularRegion	6.4.17, 6.4.18	S1.3.2.5.1:O23	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.2.	Support a rectangularRegion	6.4.17, 6.4.20	S1.3.2.5.1:O23	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.3.	Maximum number of rectangularRegions supported	6.4.17, 6.4.20	S1.3.2.5.1.2 8:M > 8:O	Enter number: ( )
S1.3.2.5.1.4.	Support a polygonalRegion	6.4.17, 6.4.21	S1.3.2.5.1:O23	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.5.	Maximum number of points in a polygonalRegion	6.4.17, 6.4.21	S1.3.2.5.1.4 8:M > 8:O	Enter number: ( )
S1.3.2.5.1.6.	Support identifiedRegion	6.4.17, 6.4.22	S1.3.2.5.1 8:M > 8:O	Enter number: ( )

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Support</b>
S1.3.2.5.1.6.1.	Maximum number of identifiedRegions supported	6.4.17, 6.4.22	S1.3.2.5.1.6: 8:M >8:O	Enter number: ( )
S1.3.2.5.1.6.2.	Support IdentifiedRegion of type CountryOnly	6.4.22, 6.4.23	S1.3.2.5.1.6:O24	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.6.3.	Support IdentifiedRegion of type CountryAndRegions	6.4.22, 6.4.24	S1.3.2.5.1.6:O24	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.6.4.	Support IdentifiedRegion of type CountryAndSubregions	6.4.22, 6.4.25	S1.3.2.5.1.6:O24	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.1.7.	Maximum number of identifiedRegions supported	6.4.17, 6.4.22	S1.3.2.5.1.6: 8:M >8:O	Enter number: ( )
S1.3.2.5.2.	Reject data if the certificate does not have the proper appPermissions	6.4.8, 6.4.28	S1.3.2.5:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.5.3.	Maximum number of PsidSsp in the appPermissions sequence	6.4.8, 6.4.28	S1.3.2.5: 8:O >8:O	Enter number: ( )
S1.3.2.5.4.	Determine that the assuranceLevel is an acceptable level	6.4.8, 6.4.27	S1.3.2.5:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.6.	Support verifying SPDUs signed with explicit authorization certificates	6.4.5	S1.3.2:O25	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.7.	Support verifying SPDUs signed with implicit authorization certificates	5.3.2, 6.4.5	S1.3.2:O25	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.8.	Support explicit certificate authority (CA) certificates	6.4.2, 6.4.6	S1.3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.9.	Support receiving implicit CA certificates	6.4.2, 6.4.5	S1.3.2:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.	SignedData verification fails in the following circumstances:	6.3.4	S1.3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.1.	... SPDU-Parsing: Invalid Input	6.3.4	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.2.	... SPDU-Parsing: Unsupported critical information field	6	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.3.	... SPDU-Parsing: Certificate not found	4.3, 6.3.13, 6.3.14, 6.3.15	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.4.	... SPDU-Parsing: Generation time not available	4.3, 6.3.13, 6.3.14, 6.3.15	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.5.	... SPDU-Parsing: Generation location not available	4.3, 6.3.13, 6.3.14, 6.3.15	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.6.	... SPDU-Certificate-Chain: Not enough information to construct chain	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.7.	... SPDU-Certificate-Chain: Chain ended at untrusted root	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.8.	... SPDU-Certificate-Chain: Chain was too long for implementation	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.9.	... SPDU-Certificate-Chain: Certificate revoked	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.10.	... SPDU-Certificate-Chain: Overdue CRL	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.11.	... SPDU-Certificate-Chain: Inconsistent expiry times	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Support</b>
S1.3.2.10.12.	... SPDU-Certificate-Chain: Inconsistent start times	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.13.	... SPDU-Certificate-Chain: Inconsistent chain permissions	5.1.2	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.14.	... SPDU-Crypto: Verification failure	5.3.1	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.15.	... SPDU-Consistency: Future certificate at generation time	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.16.	... SPDU-Consistency: Expired certificate at generation time	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.17.	... SPDU-Consistency: Expiry date too early	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.18.	... SPDU-Consistency: Expiry date too late	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.19.	... SPDU-Consistency: Generation location outside validity region	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.20.	... SPDU-Consistency: Unauthorized PSID	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.21.	... SPDU-Internal-Consistency: Expiry time before generation time	6.4.8, 6.4.14, 5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.22.	... SPDU-Internal-Consistency: extDataHash doesn't match	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.23.	... SPDU-Local-Consistency: PSIDs don't match	5.2.3	S1.3.2.10:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.24.	... SPDU-Local-Consistency: Chain was too long for SDEE	5.2.3	S1.3.2.10:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.25.	... SPDU-Relevance: SPDU Too Old	5.2.4	S1.3.2.10:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.26.	... SPDU-Relevance: Future SPDU	5.2.4	S1.3.2.10:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.27.	... SPDU-Relevance: Expired SPDU	5.2.4	S1.3.2.10:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.28.	... SPDU-Relevance: SPDU Too Distant	5.2.4	S1.3.2.10:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.2.10.29.	... SPDU-Relevance: Replayed SPDU	5.2.4	S1.3.2.10:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.	Decrypt Ieee1609Dot2Data containing EncryptedData	4.2.2.3.3, 5.3.5, 6.3.30	S1.3:O17	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.1.	Generate ECIES keypairs using a high-quality random number generator	5.3.4, 5.3.5, 5.3.6	S1.3.3: M	
S1.3.3.2.	Maximum number of RecipientInfos supported in an incoming EncryptedData	6.3.30	S1.3.3: 8:M > 8:O	Enter number: ( )
S1.3.3.2.1.	Containing symmRecipientInfo	6.3.31, 6.3.1	S1.3.3.2:O26	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.2.2.	Containing certRecipientInfo	6.3.31, 6.3.3	S1.3.3.2:O26	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.2.3.	Containing signedDataRecipientInfo	6.3.31, 6.3.3	S1.3.3.2:O26	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.2.4.	Containing rekRecipientInfo	6.3.31, 6.3.3	S1.3.3.2:O26	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.3.	Support decrypting using a public-key algorithm	6.3.5	S1.3.3:O27	<input type="checkbox"/> Yes <input type="checkbox"/> No

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Support</b>
S1.3.3.3.1.	... using ECIES-256	6.3.5	S1.3.3.3:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.3.1.1.	.... using NIST p256	6.3.5	S1.3.3.3:O28	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.3.1.2.	.... using Brainpool p256r1	6.3.5	S1.3.3.3:O28	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.3.2.	... using a different algorithm introduced at a later date	6.3.6	S1.3.3.3:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.4.	Support decrypting using a symmetric algorithm	6.3.7	S1.3.3:O27	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.4.1.	... using AES-128	6.3.7	S1.3.3.4:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S1.3.3.4.2.	... using a different algorithm introduced at a later date	6.3.3	S1.3.3.4:O	<input type="checkbox"/> Yes <input type="checkbox"/> No

### A.2.3.2 Certificate revocation list (CRL) verification entity

<b>Item</b>	<b>Security configuration (top-level)</b>	<b>Reference</b>	<b>Status</b>	<b>Suppprt</b>
S2.	Support CRL Validation Entity	7	O1	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.	Correctly verify received CRL	7.4	S2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.1.	...using hash ID-based revocation	5.1.3.5	S2.1:O29	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.1.1.	... of type fullHashCrl	7.3.2	S2.1.1:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.1.2.	... of type deltaHashCrl	7.3.2	O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.2.	... using linkage-based revocation	5.1.3.4	S2.1:O29	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.2.1.	... of type fullLinkedCrl	7.3.2	S2.1.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.2.2.	... of type deltaLinkedCrl	7.3.2	O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.2.3.	... containing individual linkage values	7.3.6	S2.1.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S2.1.2.4.	... containing group linkage values	7.3.6	O	<input type="checkbox"/> Yes <input type="checkbox"/> No

### A.2.3.3 Peer-to-peer certificate distribution (P2PCD) functionality

Item	Security configuration (top-level)	Reference	Status	Support
S3.	Support P2PCD	8	O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.1.	Number of supported SDEEs	8.2.6	S3.2: 1:O >1:O	Enter number: ( )
S3.2.	<b>Support SSME and SDS operations for P2PCD in the requester role</b>	8.2.4.1	S3:O30	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.2.1.	Under at least one condition, trigger request processing on receiving a trigger SPDU	8.2.4.1	S3.2:M	Enter description of at least one condition under which request processing is triggered ( )
S3.2.2.	Do not trigger request processing on receiving a trigger SPDU for which a request is already active	8.2.4.1	S3.2:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.2.3.	Number of simultaneously active P2PCD learning requests	8.2.4.1, 8.2.6	S3.2: 1:O >1:O	Enter number: ( )
S3.2.4.	When request processing is triggered, include a P2PCD learning request in the next SPDU for the trigger SDEE except in the following exception cases	8.2.4.1	S3.2: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.2.4.1.	Do not include a P2PCD learning request if a learning request for the same certificate has been received within <b>p2pcd_observedRequestTimeout</b>	8.2.4.1	S3.2.4:O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.2.4.2.	Only include one P2PCD learning request no matter how many learning requests have been triggered	8.2.4.1	S3.2.4: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.2.5.	Receive notifications from a P2PCDE that a P2PCD learning response has been received and use those to update the list of known certificates.	8.2.4.1	S3.2: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.3.	<b>Support SSME and SDS operations for P2PCD in the responder role</b>	8.2.4.2	S3:O30	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.3.1.	Trigger response processing on receiving a P2PCD learning request	8.2.4.2	S3.3:M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.3.2.	Number of simultaneously active P2PCD learning responses	8.2.4.1, 8.2.6	S3.3: 1:O >1:O	Enter number: ( )
S3.3.3.	Do not trigger response processing if less than <b>p2pcd_responseActiveTimeout</b> has passed since last triggered	8.2.4.2	S3.3: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.3.4.	Trigger sending response after random backoff time unless threshold number of responses have been observed	8.2.4.2	S3.3: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.3.5.	Increment number of responses observed based on input from P2PCDE	8.2.4.2	S3.3: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.4.	<b>Support P2PCDE operations for P2PCD</b>	8.2.4.2	S3:O30	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.4.1.	Receive responses and provide to SSME	8.2.4.1, 8.2.4.2, 8.3.1	S3.4: M	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.4.2.	Send responses when triggered by SSME	8.2.4.2, 8.3.1	S3.4: O	<input type="checkbox"/> Yes <input type="checkbox"/> No
S3.4.3.	Send responses over WSMP	8.2.4.2	S3.4.2: M	<input type="checkbox"/> Yes <input type="checkbox"/> No

## Annex B

(normative)

### ASN.1 modules

#### B.1 1609.2 security services

##### B.1.1 1609.2 schema

```
IEEE1609dot2 {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) base (1) schema (1)}

--
***** IEEE P1609.2 Data Types *****
--
DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
    CrlSeries,
    EccP256CurvePoint,
    EciesP256EncryptedKey,
    EncryptionKey,
    GeographicRegion,
    GroupLinkageValue,
    HashAlgorithm,
    HashedId3,
    HashedId8,
    HashedId32,
    Hostname,
    IValue,
    LinkageValue,
    Opaque,
    Psid,
    PsidSsp,
    PsidSspRange,
    PublicEncryptionKey,
    PublicVerificationKey,
    SequenceOfPsidSsp,
    SequenceOfPsidSspRange,
    ServiceSpecificPermissions,
    Signature,
    SubjectAssurance,
    SymmetricEncryptionKey,
    ThreeDLocation,
    Time64,
    Uint3,
    Uint8,
    Uint16,
```

```
    Uint32,
    ValidityPeriod
FROM IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)
    standards-association-numbered-series-standards(2) wave-stds(1609)
    dot2(2) base(1) base-types(2)}

;

--*****
-- Structures for describing secured data
--*****
-- this structure belongs later in the file but putting it here avoids
-- compiler errors with certain tools
SignedDataPayload ::= SEQUENCE {
    data           Ieee1609Dot2Data OPTIONAL,
    extDataHash    HashedData OPTIONAL,
    ...
}
(WITH COMPONENTS {..., data PRESENT} |
 WITH COMPONENTS {..., extDataHash PRESENT})

Ieee1609Dot2Data ::= SEQUENCE {
    protocolVersion   Uint8(3),
    content          Ieee1609Dot2Content
}

Ieee1609Dot2Content ::= CHOICE {
    unsecuredData      Opaque,
    signedData         SignedData,
    encryptedData     EncryptedData,
    signedCertificateRequest Opaque,
    ...
}

SignedData ::= SEQUENCE {
    hashId          HashAlgorithm,
    tbsData         ToBeSignedData,
    signer          SignerIdentifier,
    signature        Signature
}

SignerIdentifier ::= CHOICE {
    digest          HashedId8,
    certificate     SequenceOfCertificate,
    self            NULL,
    ...
}

ToBeSignedData ::= SEQUENCE {
    payload          SignedDataPayload,
    headerInfo       HeaderInfo
}

HashedData ::= CHOICE {
    sha256HashedData OCTET STRING (SIZE(32)),
    ...
}

HeaderInfo ::= SEQUENCE {
```

```

psid                      Psid,
generationTime             Time64 OPTIONAL,
expiryTime                Time64 OPTIONAL,
generationLocation         ThreeDLocation OPTIONAL,
p2pcdLearningRequest      HashedId3 OPTIONAL,
missingCrlIdentifier     MissingCrlIdentifier OPTIONAL,
encryptionKey             EncryptionKey OPTIONAL,
...
}

MissingCrlIdentifier ::= 
SEQUENCE { cracaId          HashedId3,
           crlSeries        CrlSeries,
           ...
         }

Countersignature ::= Ieee1609Dot2Data (WITH COMPONENTS {..., 
content (WITH COMPONENTS {..., 
signedData (WITH COMPONENTS {..., 
tbsData (WITH COMPONENTS {..., 
payload (WITH COMPONENTS {..., 
data ABSENT,
extDataHash PRESENT
}),
headerInfo(WITH COMPONENTS {..., 
generationTime PRESENT,
expiryTime ABSENT,
generationLocation ABSENT,
p2pcdLearningRequest ABSENT,
missingCrlIdentifier ABSENT,
encryptionKey ABSENT
})
})
})
})
})
})

--*****
-- Structures for describing encrypted data
--*****
EncryptedData ::= 
SEQUENCE { recipients       SequenceOfRecipientInfo,
           ciphertext       SymmetricCiphertext
         }

RecipientInfo ::= 
CHOICE { pskRecipInfo      PreSharedKeyRecipientInfo,
          symmRecipInfo    SymmRecipientInfo,
          certRecipInfo    PKRecipientInfo,
          signedDataRecipInfo PKRecipientInfo,
          rekRecipInfo     PKRecipientInfo
        }

SequenceOfRecipientInfo ::= SEQUENCE OF RecipientInfo

PreSharedKeyRecipientInfo ::= HashedId8
SymmRecipientInfo ::= SEQUENCE {

```

```

recipientId      HashedId8,
encKey          SymmetricCiphertext
}

PKRecipientInfo ::= SEQUENCE {
  recipientId      HashedId8,
  encKey          EncryptedDataEncryptionKey
}

EncryptedDataEncryptionKey ::= CHOICE {
  eciesNistP256      EciesP256EncryptedKey,
  eciesBrainpoolP256rl  EciesP256EncryptedKey,
  ...
}

SymmetricCiphertext ::= CHOICE { aes128ccm      AesCcmCiphertext,
  ...
}

AesCcmCiphertext ::= SEQUENCE { nonce          OCTET STRING (SIZE (12)),
  ccmCiphertext    Opaque -- 16 bytes longer than plaintext
}

--*****
-- Certificates and other security management data structures
--*****

-- Certificates are implicit (type = implicit, toBeSigned includes
-- reconstruction value, signature absent) or explicit (type = explicit,
-- toBeSigned includes verification key, signature present).

Certificate ::= CertificateBase (ImplicitCertificate | ExplicitCertificate)

SequenceOfCertificate ::= SEQUENCE OF Certificate

CertificateBase ::= SEQUENCE { version      Uint8(3),
  type          CertificateType,
  issuer        IssuerIdentifier,
  toBeSigned    ToBeSignedCertificate,
  signature     Signature OPTIONAL
}

CertificateType ::= ENUMERATED {
  explicit,
  implicit,
  ...
}

ImplicitCertificate ::= CertificateBase (WITH COMPONENTS {...,  

  type(implicit),  

  toBeSigned(WITH COMPONENTS {...,  

    verifyKeyIndicator(WITH COMPONENTS {reconstructionValue})  

  }),  

  signature ABSENT
})

```

```

ExplicitCertificate ::= CertificateBase (WITH COMPONENTS { ... ,
    type(explicit),
    toBeSigned(WITH COMPONENTS { ... ,
        verifyKeyIndicator(WITH COMPONENTS { verificationKey })
    }),
    signature PRESENT
})

IssuerIdentifier ::= CHOICE {
    sha256AndDigest      HashedId8,
    self                  HashAlgorithm,
    ...
}

ToBeSignedCertificate ::= SEQUENCE {
    id                  CertificateId,
    cracaId            HashedId3,
    crlSeries          CrlSeries,
    validityPeriod     ValidityPeriod,
    region              GeographicRegion OPTIONAL,
    assuranceLevel     SubjectAssurance OPTIONAL,
    appPermissions      SequenceOfPsidSsp OPTIONAL,
    certIssuePermissions SequenceOfPsidGroupPermissions OPTIONAL,
    certRequestPermissions SequenceOfPsidGroupPermissions OPTIONAL,
    canRequestRollover   NULL OPTIONAL,
    encryptionKey       PublicEncryptionKey OPTIONAL,
    verifyKeyIndicator   VerificationKeyIndicator,
    ...
}
(WITH COMPONENTS { ..., appPermissions PRESENT} |
 WITH COMPONENTS { ..., certIssuePermissions PRESENT} |
 WITH COMPONENTS { ..., certRequestPermissions PRESENT})

CertificateId ::= CHOICE {
    linkageData          LinkageData,
    name                 Hostname,
    binaryId             OCTET STRING(SIZE(1..64)),
    none                 NULL,
    ...
}

LinkageData ::= 
SEQUENCE { iCert           IValue,
            linkage-value  LinkageValue,
            group-linkage-value GroupLinkageValue OPTIONAL
        }

EndEntityType ::= BIT STRING {app (0), enrol (1) } (SIZE (8))

PsidGroupPermissions ::= SEQUENCE {
    appPermissions      SubjectPermissions,
    minChainDepth      INTEGER DEFAULT 1, -- 0 for enrolment certs
    chainDepthRange    INTEGER DEFAULT 0, -- max depth = min + range
    eeType              EndEntityType DEFAULT {app}
}

SequenceOfPsidGroupPermissions ::= SEQUENCE OF PsidGroupPermissions

SubjectPermissions ::= CHOICE {
    explicit            SequenceOfPsidSspRange,
    all                 NULL,
    ...
}

```

```
VerificationKeyIndicator ::= CHOICE {
    verificationKey           PublicVerificationKey,
    reconstructionValue       EccP256CurvePoint,
    ...
}
```

END

## B.1.2 1609.2 base types

```
-- -----
-- Time
--
-- -----
Time32 ::= Uint32
Time64 ::= Uint64

ValidityPeriod ::= SEQUENCE {
    start            Time32,
    duration         Duration
}

Duration ::= CHOICE {
    microseconds     Uint16,
    milliseconds    Uint16,
    seconds          Uint16,
    minutes          Uint16,
    hours            Uint16,
    sixtyHours       Uint16,
    years            Uint16
}

-- -----
-- Location
--
-- -----
```

GeographicRegion ::=  
CHOICE { circularRegion CircularRegion,  
 rectangularRegion SequenceOfRectangularRegion,  
 polygonalRegion PolygonalRegion,  
 identifiedRegion SequenceOfIdentifiedRegion,  
 ...  
}

CircularRegion ::=  
SEQUENCE { center TwoDLocation,  
 radius Uint16  
 }

RectangularRegion ::=  
SEQUENCE { northWest TwoDLocation,  
 southEast TwoDLocation  
 }

SequenceOfRectangularRegion ::= SEQUENCE OF RectangularRegion

PolygonalRegion ::= SEQUENCE SIZE(3..MAX) OF TwoDLocation

TwoDLocation ::=  
SEQUENCE { latitude Latitude,  
 longitude Longitude  
 }

IdentifiedRegion ::=  
CHOICE { countryOnly CountryOnly,  
 countryAndRegions CountryAndRegions,  
 countryAndSubregions CountryAndSubregions,  
 ...  
 }

```
}
```

SequenceOfIdentifiedRegion ::= SEQUENCE OF IdentifiedRegion

CountryOnly ::= Uint16

CountryAndRegions ::= SEQUENCE {  
 countryOnly CountryOnly,  
 regions SequenceOfUint8  
}

CountryAndSubregions ::= SEQUENCE {  
 country CountryOnly,  
 regionAndSubregions SequenceOfRegionAndSubregions  
}

RegionAndSubregions ::= SEQUENCE {  
 region Uint8,  
 subregions SequenceOfUint16  
}

SequenceOfRegionAndSubregions ::= SEQUENCE OF RegionAndSubregions

ThreeDLocation ::=  
 SEQUENCE { latitude Latitude,  
 longitude Longitude,  
 elevation Elevation  
 }

Latitude ::= NinetyDegreeInt

Longitude ::= OneEightyDegreeInt

Elevation ::= ElevInt

NinetyDegreeInt ::= INTEGER {  
 min (-900000000),  
 max (900000000),  
 unknown (900000001)  
} (-900000000..900000001)

KnownLatitude ::= NinetyDegreeInt (min..max) -- Minus 90deg to +90deg in  
microdegree intervals

UnknownLatitude ::= NinetyDegreeInt (unknown)

OneEightyDegreeInt ::= INTEGER {  
 min (-1799999999),  
 max (1800000000),  
 unknown (1800000001)  
} (-1799999999..1800000001)

KnownLongitude ::= OneEightyDegreeInt (min..max)

UnknownLongitude ::= OneEightyDegreeInt (unknown)

ElevInt ::= Uint16 -- Range is from -4096 to 61439 in units of one-tenth of a  
meter

---

-- -----  
--  
-- Crypto  
--  
-- -----

Signature ::=  
 CHOICE { ecdsaNistP256Signature EcdsaP256Signature,

```
ecdsaBrainpoolP256r1Signature EcdsaP256Signature,  
    ...  
}  
  
EcdsaP256Signature ::=  
SEQUENCE { r      EccP256CurvePoint,  
           s      OCTET STRING (SIZE (32))  
         }  
  
EccP256CurvePoint ::=  
CHOICE { x-only          OCTET STRING (SIZE (32)),  
        fill            NULL, -- consistency with 1363 / x9.62  
        compressed-y-0   OCTET STRING (SIZE (32)),  
        compressed-y-1   OCTET STRING (SIZE (32)),  
        uncompressed    SEQUENCE {  
                                x OCTET STRING (SIZE (32)),  
                                y OCTET STRING (SIZE (32))  
                            }  
         }  
  
SymmAlgorithm ::= ENUMERATED {  
    aes128Ccm,  
    ...  
}  
  
HashAlgorithm ::= ENUMERATED {  
    sha256,  
    ...  
}  
  
EciesP256EncryptedKey ::=  
SEQUENCE { v      EccP256CurvePoint,  
           c      OCTET STRING (SIZE (16)),  
           t      OCTET STRING (SIZE (16))  
         }  
  
EncryptionKey ::= CHOICE {  
    public          PublicEncryptionKey,  
    symmetric       SymmetricEncryptionKey  
}  
  
PublicEncryptionKey ::= SEQUENCE {  
    supportedSymmAlg   SymmAlgorithm,  
    publicKey         BasePublicEncryptionKey  
}  
  
BasePublicEncryptionKey ::= CHOICE {  
    eciesNistP256     EccP256CurvePoint,  
    eciesBrainpoolP256r1   EccP256CurvePoint,  
    ...  
}  
  
PublicVerificationKey ::= CHOICE {  
    ecdsaNistP256     EccP256CurvePoint,  
    ecdsaBrainpoolP256r1   EccP256CurvePoint,  
    ...  
}  
  
SymmetricEncryptionKey ::= CHOICE {  
    aes128Ccm        OCTET STRING(SIZE(16)),  
    ...  
}
```

```
-- -----
-- PSID / ITS-AID
--



PsidSsp ::= SEQUENCE {
    psid                  Psid,
    ssp                   ServiceSpecificPermissions OPTIONAL
}

SequenceOfPsidSsp ::= SEQUENCE OF PsidSsp

Psid ::= INTEGER (0..MAX)

SequenceOfPsid ::= SEQUENCE OF Psid

ServiceSpecificPermissions ::= CHOICE {
    opaque                OCTET STRING (SIZE(0..MAX)),
    ... -- will define Structured SSP here
}

PsidSspRange ::= SEQUENCE {
    psid                  Psid,
    sspRange              SspRange OPTIONAL
}

SequenceOfPsidSspRange ::= SEQUENCE OF PsidSspRange

SspRange ::= CHOICE {
    opaque                SequenceOfOctetString,
    all                  NULL,
    ... -- will define Structured SSP Range here
}

SequenceOfOctetString ::= SEQUENCE (SIZE (0..MAX)) OF
    OCTET STRING (SIZE(0..MAX))

-- -----
-- Goes in certs
--



SubjectAssurance ::= OCTET STRING (SIZE(1))

CrlSeries ::= Uint16

-- -----
-- Pseudonym Linkage
--



IValue ::= Uint16
Hostname ::= UTF8String (SIZE(0..255))
LinkageValue ::= OCTET STRING (SIZE(9))
GroupLinkageValue ::= SEQUENCE {
```

```
jValue    OCTET STRING (SIZE(4)),  
value     OCTET STRING (SIZE(9))  
}  
  
LaId ::= OCTET STRING (SIZE(2))  
LinkageSeed ::= OCTET STRING (SIZE(16))  
  
END
```

## B.2 Certificate revocation list (CRL)

### B.2.1 Certificate revocation list: Base types

```
IEEE1609dot2CrlBaseTypes {iso(1) identified-organization(3) ieee(111)  
standards-association-numbered-series-standards(2) wave-stds(1609)  
dot2(2) crl(3) base-types(2)}  
  
DEFINITIONS AUTOMATIC TAGS ::= BEGIN  
  
EXPORTS ALL;  
  
IMPORTS  
    CrlSeries,  
    GeographicRegion,  
    HashedId8,  
    HashedId10,  
    IValue,  
    LaId,  
    LinkageSeed,  
    Opaque,  
    Psid,  
    Signature,  
    Time32,  
    Uint3,  
    Uint8,  
    Uint16,  
    Uint32,  
    ValidityPeriod  
FROM IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)  
standards-association-numbered-series-standards(2) wave-stds(1609)  
dot2(2) base(1) base-types(2)}  
;  
  
--  
--  
-- CRL contents  
--  
--  
  
CrlContents ::= SEQUENCE {  
    version          Uint8 (1),  
    crlSeries       CrlSeries,  
    cracaId        HashedId8,  
    issueDate      Time32,  
    nextCrl        Time32,  
    priorityInfo   CrlPriorityInfo,  
    typeSpecific   CHOICE {  
        fullHashCrl   ToBeSignedHashIdCrl,
```

```

        deltaHashCrl          ToBeSignedHashIdCrl,
        fullLinkedCrl         ToBeSignedLinkageValueCrl,
        deltaLinkedCrl        ToBeSignedLinkageValueCrl,
        ...
    }
}

CrlPriorityInfo ::= SEQUENCE {
    priority              Uint8 OPTIONAL,
    ...
}

ToBeSignedHashIdCrl ::= SEQUENCE {
    crlSerial             Uint32,
    entries               SequenceOfHashBasedRevocationInfo,
    ...
}

HashBasedRevocationInfo ::= SEQUENCE {
    id                   HashedId10,
    expiry                Time32
}

SequenceOfHashBasedRevocationInfo ::= SEQUENCE OF HashBasedRevocationInfo

ToBeSignedLinkageValueCrl ::= SEQUENCE {
    iRev                  IValue,
    indexWithinI           Uint8,
    individual             SequenceOfJMaxGroup OPTIONAL,
    groups                 SequenceOfGroupCrlEntry OPTIONAL,
    ...
}
(WITH COMPONENTS {..., individual PRESENT} |
 WITH COMPONENTS {..., groups PRESENT})

JMaxGroup ::= SEQUENCE {
    jmax                  Uint8,
    contents              SEQUENCE OF LAGroup
}

SequenceOfJMaxGroup ::= SEQUENCE OF JMaxGroup

LAGroup ::= SEQUENCE {
    la1Id                 LaId,
    la2Id                 LaId,
    contents              SequenceOfIMaxGroup
}

SequenceOfLAGroup ::= SEQUENCE OF LAGroup

IMaxGroup ::= SEQUENCE {
    iMax                  Uint16,
    contents              SequenceOfIndividualRevocation
}

SequenceOfIMaxGroup ::= SEQUENCE OF IMaxGroup

IndividualRevocation ::= SEQUENCE {

```

```
linkage-seed1      LinkageSeed,
linkage-seed2      LinkageSeed
}

SequenceOfIndividualRevocation ::= SEQUENCE OF IndividualRevocation

GroupCrlEntry ::= SEQUENCE {
    iMax              Uint16,
    la1Id            LaId,
    linkageSeed1     LinkageSeed,
    la2Id            LaId,
    linkageSeed2     LinkageSeed
}

SequenceOfGroupCrlEntry ::= SEQUENCE OF GroupCrlEntry

END
```

## B.2.2 CRL: Security envelope

```
IEEE1609dot2Crl {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) crl(3) protocol(1)}

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS

    Ieee1609Dot2Data
FROM IEEE1609dot2 {iso(1) identified-organization(3) ieee(111)
    standards-association-numbered-series-standards(2) wave-stds(1609)
    dot2(2) base(1) schema(1)}

    Psid,
    Opaque
FROM IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)
    standards-association-numbered-series-standards(2) wave-stds(1609)
    dot2(2) base(1) base-types(2)}

    CrlContents
FROM IEEE1609dot2CrlBaseTypes {iso(1) identified-organization(3) ieee(111)
    standards-association-numbered-series-standards(2) wave-stds(1609)
    dot2(2) crl(3) base-types(2)}

;

CrlPsid ::= Psid(135)

SecuredCrl ::= Ieee1609Dot2Data (WITH COMPONENTS {...,
    content (WITH COMPONENTS {
        signedData (WITH COMPONENTS {...,
            tbsData (WITH COMPONENTS {
                payload (WITH COMPONENTS {...,
                    data (WITH COMPONENTS {...,
                        content (WITH COMPONENTS {
                            unsecuredData (CONTAINING CrlContents)
                        })
                    })
                })
            })
        })
    })
},
```

```

        headerInfo (WITH COMPONENTS {....,
            psid (CrlPsid),
            generationTime ABSENT,
            expiryTime ABSENT,
            generationLocation ABSENT,
            p2pcdLearningRequest ABSENT,
            missingCrlIdentifier ABSENT,
            encryptionKey ABSENT
        })
    })
}
}
}

END

```

### B.2.3 CRL: Service Specific Permissions (SSP)

```

IEEE1609dot2CrlSsp {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) crl(3) service-specific-permissions (3)}

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
    CrlSeries,
    Uint8
FROM IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) base(1) base-types(2)}
;

CrlSsp ::= SEQUENCE {
    version          Uint8(1),
    associatedCraca CracaType,
    crls             PermissibleCrls,
    ...
}

CracaType ::= ENUMERATED {isCraca, issuerIsCraca}

PermissibleCrls ::= SEQUENCE OF CrlSeries

END

```

### B.3 Peer-to-peer certificate distribution (P2PCD)

```

IEEE1609dot2-Peer2Peer {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) management (2) peer-to-peer (1)}

--*****  

--  

-- Data types for Peer-to-peer distribution of IEEE P1609.2 support data

```

```
--  
-- Associated with a two-byte PSID to be assigned.  
-- When broadcast over WSMP, to be encoded with COER.  
--  
--*****  
DEFINITIONS AUTOMATIC TAGS ::= BEGIN  
EXPORTS ALL;  
IMPORTS  
    Uint8  
FROM IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)  
    standards-association-numbered-series-standards(2) wave-stds(1609)  
    dot2(2) base(1) base-types(2)}  
  
    Certificate  
FROM IEEE1609dot2 {iso(1) identified-organization(3) ieee(111)  
    standards-association-numbered-series-standards(2) wave-stds(1609)  
    dot2(2) base(1) schema(1)}  
;  
  
Ieee1609dot2Peer2PeerPDU ::= SEQUENCE {  
    version          Uint8(1),  
    content          CHOICE {  
        caCerts       CaCertP2pPDU,  
        ...  
    }  
}  
  
CaCertP2pPDU ::= SEQUENCE OF Certificate  
END
```

## Annex C

(informative)

### Specifying the use of IEEE Std 1609.2 by SDEEs

#### C.1 General

A complete specification of a secure data exchange entity (SDEE), i.e., an entity that uses Wireless Access in Vehicular Environments secure data service (WAVE SDS), includes a specification of how WAVE SDS is used. This annex provides information to assist the SDEE specifier in providing that specification, in the form of:

- The 1609.2 security profile, which defines output data structures as well as how optional aspects of interaction between the SDEE and the SDS are to be carried out: see C.2.
- An overview of Service Specific Permissions (SSPs) for the SDEE and how they map to permitted payloads: see C.4.
- An overview of additional restrictions on the certificates used by the SDEE: see C.6.

An organization that specifies a SDEE may wish to provide minimum requirements for performance for an implementation of that entity. For example, there may be a required accuracy metric for the estimate of the time. These minimum performance requirements are outside the scope of this standard.

#### C.2 IEEE 1609.2 security profiles

##### C.2.1 Contents of security profile

###### C.2.1.1 General

The IEEE 1609.2 security profile for a SDEE is a compact description of the security processing that the entity carries out. It is intended for inclusion in a full specification of a SDEE, including application behavior. A security profile is linked to one or more Provider Service Identifiers (PSIDs) and specifies the security behavior of SDEEs associated with that PSID.

The IEEE 1609.2 security profile specifies the data structures that a SDEE should output. It also specifies the WAVE Security Services primitives that a calling SDEE can invoke to obtain such output and gives instructions as to how to set the values of the parameters of those primitives. The IEEE 1609.2 security profile may set these parameters to specific values or it may describe parameters as “variable”, in which case the mechanism for setting the parameter values is intended to be described in text. For each entry in a profile, the profile contains the entry name, the entry value, and notes. The notes are part of the SDEE specification and may be used to provide information beyond the information provided by the entry value.

A security profile is part of a complete specification of an application area. As such, the application area specifier has full discretion as to how to use the security profile to provide that complete specification. For example, some of the entries in the security profile may take different values under different conditions, or there may be different security profiles for the same application operating in different settings, or there may be different security profiles for different information flows within an application. Additionally, the

application area specifier may choose to use some mechanism other than the security profile to specify security operations.

The IEEE 1609.2 security profile contains four sections:

- IEEE 1609.2 security profile identification: describes the IEEE 1609.2 security profile
- Sending: describes options to be set when creating secured data for sending
- Receiving: describes options to be set when processing received secured data
- Security management: describes constraints on the certificates to be used

In the sending and receiving sections, certain entries in the security profile specify the values of particular fields within a secured protocol data unit (SPDU). A SPDU is an Ieee1609Dot2Data. Fields within the Ieee1609Dot2Data are identified using “dot notation”: for example, if a security profile entry governs the contents of the following:

- The field *data*
- Within the SignedDataPayload *payload*
- Within the ToBeSignedData *tbsData*
- Within the SignedData *signedData*
- Within the Ieee1609Dot2Content *content*
- Within an Ieee1609Dot2Data

Then “dot notation” indicates that field by denoting the Ieee1609Dot2Data by *d*, and referring to “the field *d.content.signedData.tbsData.payload.data*”.

### C.2.1.2 SDS

#### C.2.1.3 IEEE 1609.2 security profile identification

Name	Type	Recommended values	Description
<i>Name</i>	Text string	Text string	The name to be used to refer to the profile. This should be unique among names used by security profiles that reference a particular PSID.
<i>PSIDs</i>	List of PSIDs	Any list of one or more PSIDs	The PSIDs to be used by SDEEs that use this profile.
<i>Other considerations</i>	Text string	Text string	A description of the conditions under which this security profile is to be used.

**Guidance for SDEE specifiers:**

- *Other considerations*: The description provided for this entry should be as specific as possible to avoid ambiguity about how and under what circumstances the security profile is to be used.

### C.2.1.3.1 Sending

This part of the IEEE 1609.2 security profile contains the following information.

Name	Type	Recommended values	Description
<i>Sign Data</i>	enumerated	True	The entity signs all outgoing data (with Sec-SignedData.request), outputting an Ieee1609Dot2Data <i>d</i> with <i>d.content</i> indicating <i>signedData</i> .
		False	The entity does not sign outgoing data.
		Text	Provide a description of how the SDEE determines which outgoing protocol data units (PDUs) to sign.
<i>Signed Data in Payload</i>	Boolean	True	If true, in the output signed SPDU which is an Ieee1609Dot2Data <i>d</i> , the field <i>d.content.signedData.tbsData.payload.data</i> is present. If false, it is absent.
		False	
<i>External Data</i>	Boolean	True False	If true, in the output signed SPDU which is an Ieee1609Dot2Data <i>d</i> , the field <i>d.content.signedData.tbsData.payload.extDataHash</i> is present. If false, it is absent.
<i>External Data Source</i>	Text		How the SDEE obtains any data that is hashed to provide <i>extDataHash</i> .
<i>External Data Hash Algorithm</i>	enumerated	“SHA-256”	The algorithm used to hash the external data.
<i>Set Generation Time in Security Headers</i>	Boolean	True False	The value set as <i>Set Generation Time</i> when invoking Sec-SignedData.request. If True, in the output signed SPDU which is an Ieee1609Dot2Data <i>d</i> , the field <i>d.content.signedData.tbsData.headerInfo.generationTime</i> is present. If false, it is absent.
<i>Set Generation Location in Security Headers</i>	Boolean	True False	The value set as <i>Set Generation Location</i> when invoking Sec-SignedData.request. If True, in the output signed SPDU which is an Ieee1609Dot2Data <i>d</i> , the field <i>d.content.signedData.tbsData.headerInfo.generationLocation</i> is present. If false, it is absent.
<i>Set Expiry Time in Security Headers</i>	Boolean	True False	The value set as <i>Set Expiry Time</i> when invoking Sec-SignedData.request. If True, in the output signed SPDU which is an Ieee1609Dot2Data <i>d</i> , the field <i>d.content.signedData.tbsData.headerInfo.expiryTime</i> is present. If false, it is absent.
<i>Signed SPDU Lifetime</i>	Time interval	Any time interval, or “n/a” if <i>SetExpiryTimeIn-SecurityHeaders</i> is False, or “Text” if a more complicated description is required	The lifetime of a signed SPDU, i.e., the time interval between the generation time and the expiry time. Provided only if <i>Set Expiry Time in Security Headers</i> is True. In this case, the field <i>d.content.signedData.tbsData.headerInfo.expiryTime</i> takes the value (current time + <i>Signed SPDU Lifetime</i> ).
<i>Signer Identifier Policy Type</i>	Enumerated	Simple Text	Describes the type of the Signer Identifier Policy. In the output signed SPDU, which is an Ieee1609Dot2Data <i>d</i> , the Signer Identifier Policy indicates which option in the field <i>d.content.signedData.signer</i> is selected.  If this is “Simple”, the Simple Signer Identifier Policy fields below are specified. If it is “Text”, the Text Signer Identifier Policy field below is specified.

Name	Type	Recommended values	Description
<i>Simple Signer Identifier Policy: Minimum Inter Cert Time</i>	Time interval (for example, “one second”)	Any valid interval of time, or “always”	<p>Used to set <i>Signer Identifier Type</i> when invoking Sec-SignedData.request, i.e., indicates which option in the field <i>d.content.signedData.signer</i> is selected.</p> <p>If the certificate being signed with has not been attached to as signed SPDU within this time, i.e., if a sign operation has not set Signer Identifier Type to <i>certificate</i> within this time or if the certificate has not been used within this time, or if this value is “always”, Sec-SignedData.request primitive is invoked with <i>Signer Identifier Type</i> set to “<i>certificate</i>” and <i>Signer Identifier Cert Chain Length</i> set to <i>Simple Signer Identifier Policy: Cert ChainLength</i>. In terms of the output, the field <i>d.content.signedData.signer.certificate</i> is present and contains (<i>Simple Signer Identifier Policy: Cert ChainLength</i>) certificates.</p> <p>Otherwise, the Sec-SignedData.request primitive is invoked with <i>Signer Identifier Type</i> set to <i>digest</i> and in the output Ieee1609Dot2Data <i>d</i>, the field <i>d.content.signedData.signer.digest</i> is present.</p>
<i>Simple Signer Identifier Policy: Exceptions</i>	Boolean	True False	Any exceptions to the simple policy are recorded in the notes.
<i>Simple Signer Identifier Policy: Signer Identifier Cert Chain Length</i>	Integer or enumerated	-256 to -1 1 to 256 “Max”	The value set as the <i>Signer Identifier Cert Chain Length</i> when invoking Sec-SignedData.request.
<i>Text Signer Identifier Policy</i>	Text	Human-readable text	A text description of how the <i>Signer Identifier Type</i> is set, i.e., which option in the field <i>d.content.signedData.signer</i> is selected.
<i>Sign With Fast Verification</i>	enumerated	Uncompressed Compressed No Optional	<p>The value set as <i>Sign With Fast Verification</i> when invoking Sec-SignedData.request. If “optional”, implementations are allowed but not required to provide fast verification data. If “No”, an implementation that provides fast verification data is not conformant.</p> <p>In terms of the output Ieee1609Dot2Data <i>d</i>: if this value is “Uncompressed”, the field <i>d.content.signedData.signer.signature.[ecdsa256signature ecdsaBrainpoolP256r1Signature].r</i> indicates <i>uncompressed</i>; if this value is “compressed”, that field indicates <i>compressed-y-0</i> or <i>compressed-y-1</i>; if it is “no”, that field indicates <i>x-only</i>; if it is “optional”, the field may indicate any of the choices.</p>
<i>EC Point Format</i>	Enumerated	Uncompressed Compressed Variable	<p>The value set as the <i>EC Point Format</i> when invoking Sec-SignedData.request.</p> <p>In terms of the output Ieee1609Dot2Data <i>d</i>: if this is “Uncompressed”, then any elliptic curve point fields in <i>d</i> indicate the choice <i>uncompressed</i>; if this is “Compressed”, then any elliptic curve point fields in <i>d</i> indicate the choice <i>compressed-y-0</i> or <i>compressed-y-1</i>.</p>
<i>p2pcd_useInteractive-Form</i>	Boolean		Whether to use the peer-to-peer certificate distribution defined in Clause 8.
<i>p2pcd_max-ResponseBackoff</i>	Time or n/a		If <i>p2pcduseInteractiveForm</i> is True, the maximum backoff time when responding to a request as defined in Clause 8. Otherwise, “n/a”.

Name	Type	Recommended values	Description
<i>p2pcd_response-ActiveTimeout</i>	Time or n/a		If <i>p2pcduseInteractiveForm</i> is True, the time after which a response-active state ends with respect to a particular trigger certificate as defined in Clause 8. Otherwise, “n/a”.
<i>p2pcd_request-ActiveTimeout</i>	Time or n/a		If <i>p2pcduseInteractiveForm</i> is True, the time before which a second request will not be sent for a particular certificate after sending the first request, as defined in Clause 8. Otherwise, “n/a”.
<i>p2pcd_observed-RequestTimeout</i>	Time or n/a		If <i>p2pcduseInteractiveForm</i> is True, the time before which a request will not be sent for a particular certificate after observing the another party’s request for the same certificate, as defined in Clause 8. Otherwise, “n/a”.
<i>p2pcd_currentlyUsed-TriggerCertificate-Time</i>	Time or n/a		If <i>p2pcduseInteractiveForm</i> is True, a time used to determine whether a trigger certificate is “currently used” as defined in Clause 8. Otherwise, “n/a”.
<i>p2pcd_response-CountThreshold</i>	Integer or n/a		If <i>p2pcduseInteractiveForm</i> is True, a number used to determine whether or not a response is sent to a particular P2PCD request as defined in Clause 8. Otherwise, “n/a”.
<i>Repeat Signed SPDUs</i>	Boolean	True False	Whether each new PDU is signed or a signed SPDU is repeated for its lifetime.
<i>Time Between Signing</i>	Time or n/a		If <i>Repeat Signed SPDUs</i> is True, the time between generating fresh signed SPDUs.
<i>Encrypt Data</i>	enumerated	No Text	Specifies whether encryption is used, and if so how the entity obtains the encryption key. “Text” indicates that a full text description is provided.

#### Guidance for SDEE specifiers:

- *Sign Data*: If the data is signed only under certain circumstances, indicate what those circumstances are. If the circumstances are consistent (for example, certain information flows are signed and certain ones aren’t), consider specifying two different security profiles for the same flow.
- *Signed Data in Payload*: In general, including the signed data in the payload is the most robust solution and should be preferred.
- *External Data, External Data Source*: If external signed data is used, the notes section should indicate unambiguously how it is obtained and how it is formatted for hashing. One possible use of external data would be to provide assurance that a given instance of signed data is associated with a session, by providing the session ID or the hash of all previous session traffic as the external data. This field should be used to specify both how the sender and the receiver obtain the external data.
- *External Data Hash*: Only SHA-256 is supported.
- *Set Generation Time in Security Headers*: The SDEE should set generation time in the security headers if it is not included in the signed SPDU payload, or if it is included in the signed SPDU payload but not with a long enough time counter to prevent replay attacks. This requirement arises from the need for the generation time of the message to fall during the validity period of the certificate.
- *Set Generation Location in Security Headers*: Set to “True” if the generation location is significant and if it is not given in the payload. Set to “False” if the generation location is not relevant (for example, for Certificate Revocation Lists) or if it is given in the PDU payload (for example, for typical safety messages).

- *Set Expiry Time in Security Headers, Signed PDU Lifetime:* The expiry time need not be set if there is a way for the receiving SDEE to discard too-old messages, for example:
  - The PDU processing itself rejects too-old messages.
  - There is some default lifetime such that it is always safe to reject messages older than that and always safe to pass messages less old than that to the SDEE. See for example the Basic Safety Message security profile in SAE J2945/1 [B21].
  - The PDU payload contains an expiry time.
- If the safe lifetime of two different PDUs may be significantly different in a way that the sending device can predict, but a receiving device cannot know, then an expiry time should be included. For example, in the case of the WAVE Service Advertisement (WSA), a Provider may add or remove services available at any time during working hours (and so may want to have a short WSA lifetime so that only current information is accepted) but may not update overnight (and so may be able to have a longer lifetime during those hours). The *Signed PDU Lifetime* should be the maximum time for which the SDEE specifier determines that there is no significant risk to a receiver from accepting an out-of-date SPDU.
- *Signer Identifier Policy Type:* Set to “Simple” if the policy can be stated using the simple fields, i.e., if the policy consists of sending a digest X times and a single other signer identifier type Y times during a given time period. Set to “Text” otherwise. In general, for settings where predistribution of CA certificates is possible and channel capacity is constrained, this can be set to Simple with *Simple Signer Identifier Policy: Minimum Inter Cert Time* set to about 0.5 seconds and *Simple Signer Identifier Policy: Signer Identifier Cert Chain Length* set to 1, i.e., only the end-entity certificate is ever sent. For settings where predistribution of CA certificates is not possible but channel capacity is not constrained, *Simple Signer Identifier Policy: Minimum Inter Cert Time* set to about 0.5 seconds and *Simple Signer Identifier Policy: Signer Identifier Cert Chain Length* set to -1. For other scenarios, the SDEE specifier states the best signer identifier policy. For any SDEE that uses this approach, it will attach a full certificate the first time it signs with that certificate.
- *Sign With Fast Verification:* This should in general be “Compressed”. There is no advantage to “No” over “Optional”.
- *EC Point Format:* This should be “Compressed” if channel capacity is limited, “Uncompressed” otherwise.
- *p2pcd\_useInteractiveForm* and the interactive-form *p2pcd\_\** variables: in general it is recommended that SDEEs use this. The *p2pcd\_\** variables should be set so as to manage the amount of additional data traffic on the channel caused by P2PCD. For example, if the values selected are *p2pcd\_maxResponseBackoff* = 0.25 s, *p2pcd\_responseActiveTimeout* = 0.25 s, *p2pcd\_requestActiveTimeout* = 0.25 s, *p2pcd\_observedRequestTimeout* = 0.25 s, *p2pcd\_currentlyUsedTriggerCertificateTime* = 1 minute, *p2pcd\_responseCountThreshold* = 3, then each unknown certificate adds about 12 messages per second, possibly slightly more because of hidden node effects. If *p2pcd\_requestActiveTimeout* = 0, the requesting device will send a request without regard to whether or not other devices are also requesting the same certificate.
- *Repeat Signed PDUs, Time Between Signing:* If a PDU’s contents change very slowly, it can reduce the computational burden on a sending device if the PDU is signed at time t and then retransmitted until its expiry time. It is recommended that this is only used if *Set Expiry Time in Security Headers* is true to reduce the risk that an old PDU is accepted as valid. If *Repeat Signed PDUs* is specified, *Time Between Signing* should also be specified as a time or as an algorithm used to determine the time between signing events. *Time Between Signing* should be similar to *Signed SPDU Lifetime*.
- *Encrypt Data:* A specification of a SDEE that encrypts data includes a specification of how the SDEE obtains the key(s) or certificate(s) to which the data is encrypted. Since different flows might obtain the keys in different ways, there might be different security profiles for different flows.

### C.2.1.3.2 Receiving

This part of the IEEE 1609.2 security profile contains the following information.

Name	Type	Valid range	Description
<i>Use Preprocessing</i>	Enumerated	True False Text	Specifies whether or not a receiving SDEE invokes Sec-SecureDataPreprocessing.confirm. This should be set to “False” if <i>Sign Data</i> in the sending policy is False. This should be set to “True” if the signer identifier policy in the sending profile allows a SignerIdentifier of type <i>digest</i> . It should also be set to “True” if <i>p2pcd_useInteractiveForm</i> is True in the sending profile. The “Text” option is provided in case there are conditions that should be evaluated to decide whether or not to invoke preprocessing.
<i>Verify Data</i>	Enumerated	True False Text	Specifies whether or not a receiving entity attempts to verify data.  If <i>SignData</i> in the sending profile is False, this is set to “False”.  If <i>SignData</i> in the sending profile is True, this is set to “True” to denote that the receiving entity attempts to verify all incoming data, or “Text” to denote that there is a fuller textual description.
<i>Maximum Certificate Chain Length</i>	Integer	Integer $\geq 2$	The value set as <i>Maximum Certificate Chain Length</i> when invoking Sec-SignedData.request and Sec-SignedDataVerification.request.
<i>Relevance: Replay</i>	Boolean	True False	The value set as <i>Relevance: Replay</i> when invoking Sec-SignedDataVerification.request.
<i>Relevance: Generation Time in Past</i>	Boolean	True False	The value set as <i>Relevance: Generation Time in Past</i> to Sec-SignedDataVerification.request.
<i>Validity Period</i>	Time interval	Any time interval such as “5 seconds” or “between 1 and 2 seconds”	The value to set as <i>Validity Period</i> when invoking Sec-SignedDataVerification.request. Set if <i>Relevance: Generation Time in Past</i> or <i>Relevance: Replay</i> is “True”. May be a text description rather than a single time period. This is the time interval such that signed SPDUs with a generation time in the past by more than <i>Validity Period</i> are rejected as invalid.
<i>Relevance: Generation Time in Future</i>	Boolean	True False	The value set as <i>Relevance: Generation Time in Future</i> to Sec-SignedDataVerification.request.
<i>Acceptable Future Data Period</i>	Time	Any positive time value	The value set as <i>Acceptable Future Data Period</i> when invoking Sec-SignedDataVerification.request, or the algorithm for setting that value.
<i>Generation Time Source</i>	Enumerated	Security Headers Payload	If <i>GenerationTimeSource</i> is “Security Headers”, the generation time parameters to Sec-SignedDataVerification.request is obtained from the corresponding field in the HeaderInfo of the SPDU.  If <i>GenerationTimeSource</i> is “Payload”, the generation time is obtained from elsewhere (for example, from the payload) and are provided by the entity to Sec-SignedDataVerification.request.
<i>Relevance: Expiry Time</i>	Boolean	True False	The value set as <i>Expiry Time Relevance Check</i> when invoking Sec-SignedDataVerification.request.

Name	Type	Valid range	Description
Expiry Time Source	Enumerated	This need only be set if <i>Expiry Time Relevance Check</i> is true.	
		Security Headers	The <i>Expiry Time</i> parameter to Sec-SignedDataVerification.request is obtained from the corresponding fields in Sec-SecureDataPre-processing.confirm.
		Payload	The <i>Expiry Time</i> is obtained from elsewhere (for example, from the payload) and is provided by the entity to Sec-SignedDataVerification.request.
Consistency: Generation Location	Boolean	True False	Whether the receiving SDEE should carry out consistency checks based on generation location.
Relevance: Generation Location Distance	Boolean or “Text”	True False Text	Whether or not to request the SDS to reject messages that are too far from the receiver. If “True” or “False”, this is set as <i>Reject Distant Messages</i> when invoking Sec-SignedDataVerification.request. If the decision on whether or not to request the SDS to reject messages that are too distant depends on context, then the value should be set to “Text” and the Notes column should explain how the decision is made.
Validity Distance	Distance in meters or “Variable”	Any positive distance or “Variable”	The value set as <i>Validity Distance</i> when invoking Sec-SignedDataVerification.request. Set only if <i>Reject Distant Messages</i> is “True”.
Generation Location Source	Enumerated	This is specified if <i>Reject Distant Messages</i> or <i>GenerationLocationConsistencyCheck</i> is true, or <i>SupportedGeographicRegions</i> (see C.2.1.3.3) contains any value other than “None”.	
		“Security Headers”	The <i>Generation Location</i> parameter to Sec-SignedDataVerification.request is obtained from the corresponding fields in Sec-SecureDataPre-processing.confirm.
		“Payload”	The <i>Generation Location</i> is obtained from the payload of the PDU and is provided by the entity to Sec-SignedDataVerification.request.
		“Other”	The <i>Generation Location</i> is obtained from some other source and is provided by the entity to Sec-SignedDataVerification.request. The source from which the generation location is obtained is provided as part of the SDEE specification.
Overdue CRL Tolerance	Time period or text	Any positive time period or text	The value set as <i>Overdue CRL Tolerance</i> when invoking Sec-SignedDataVerification.request.
Relevance: Certificate Expiry	Boolean	True False	Whether or not to carry out the certificate expiry relevance check specified in 5.2.4.2.7.
Accept Encrypted Data	Enumerated or text	“Exclusively”	The entity rejects any received data that are not encrypted.
		“No”	The entity rejects any received data that are encrypted.
		Text	Depending on conditions to be specified by the organization that specifies the IEEE 1609.2 security profile, the entity may accept non-encrypted data or encrypted data.

### Guidance for users:

- *Verify Data*: False if *Sign Data* is set to “False” in the profile for the incoming flow. Otherwise, should specify the criteria used to decide whether verification is necessary. An implementation of a SDEE may verify more incoming messages than the ones that meet these criteria.
- *Generation Time Relevance Check*: It is recommended that either this or expiry time is specified.
- *Generation Time Source*: This field is used to check validity of generation time against the certificate validity period, so this is necessary in a 1609.2 Security Profile even if *Generation Time Relevance Check* is false.
- *Expiry Time Relevance Check*: Consistent with *Set Expiry Time in Security Headers* in the send security profile.
- *Expiry Time Source*: Consistent with *Set Expiry Time in Security Headers* in the send security profile.
- *Reject Distant Messages*: Set to “True” if the SDS is desired to reject distant messages. Set to “False” if the SDEE rejects distant messages as part of the SDEE processing, or if generation location is irrelevant.
- *Generation Location Source*: Consistent with *Set Generation Location in Security Headers* in the send security profile.
- *Accept Encrypted Data*: Consistent with *Encrypt Data* in the send security profile.
- *Detect Replay*: Set to “True” if (a) replayed messages (i.e., the same message, acted upon twice) are a threat and (b) the SDEE processing does not automatically reject replayed messages.
- *Data Validity Period*: A reasonable lifetime for the data. For example, for time-critical safety applications, this might be a small number of seconds, while for other messages with less dynamic contents it might be longer.
- *Data Validity Distance*: A reasonable generation distance for the data. Most likely to be between 300 m and 1000 m, depending on whether the data is expected to be generated by roadside equipment (RSE) or on-board equipment (OBE).
- *Acceptable Future Data Period*: Depends on the time-sensitivity of the receiving SDEE. If the SDEE is not very time-sensitive, then data that claims to have been generated slightly in the future is acceptable. Any value greater than about 0.5 seconds requires strong justification.
- *Maximum Certificate Chain Length*: Should be consistent with *Simple Signer Identifier Policy*.
- *Signer Identifier Cert Chain Length*: Indicates how long a certificate chain is expected to be for this SDEE, and so the certificate chain length that a 1609.2 implementation should support.
- *Overdue CRL Tolerance*: This value depends on a number of factors including: (a) the likelihood that devices that implement this SDEE specification have a reliable internet data connection—higher likelihood should lower this value; (b) the risk from accepting false messages versus rejecting true messages—the greater the relative risk of false messages, the lower this value should be; (c) the typical lifetime of a collection of certificates issued for implementations of this SDEE specification; if an implementations’ certificates reach only a short time into the future, then certificate revocation lists (CRLs) are less important and the *Overdue CRL Tolerance* value can be set to be large. This value may any number from seconds to weeks or months. Additionally, if it takes the value “Text”, it may include exceptions, such as allowing a grace period if the SDEE has been inactive for a long time during which the SSME may attempt to obtain the CRL.
- *Relevance: Certificate Expiry*: It is strongly recommended that this is set to “True” unless the SDEE has assurance that expired certificates remain on the CRL for some period of time.
- *Accept Encrypted Data*: Set to “True” if the SDEE receives encrypted data.

### C.2.1.3.3 Security management

This part of the IEEE 1609.2 security profile contains the following information for each PSID for which the entity uses an Ieee1609Dot2Data structure:

Name	Type	Valid range	Description
<i>Signing Key Algorithm</i>	Enumerated	ecdsaNistP256withSha1 ecdsaBrainpoolP256r1withSha1	One of the valid signing algorithms identified in 5.3.1 and 6.4.36.
<i>Encryption Algorithm</i>	Enumerated	eciesNistP256 eciesBrainpoolP256r1	One of the valid encryption algorithms identified in 5.3.5 and 6.3.20.
<i>Implicit or Explicit Certificates</i>	Enumerated	Explicit	A receiver supports receiving explicit certificates only and a sender uses an explicit certificate only for a given transmission.
		Implicit	A receiver supports receiving implicit certificates only and a sender uses an implicit certificate only for a given transmission.
		Both	A receiver supports receiving both implicit and explicit certificates and a sender may choose to use either an implicit or an explicit certificate for a given transmission if it has certs of both types available.
<i>EC Point Format</i>	Enumerated	Compressed Uncompressed	How points are to be represented in certificates.
<i>Supported Geographic Regions</i>	Array of enumerated	An array of entries, each of which is one of:  None Rectangular Circular Polygonal  Identified: Country Only Identified: Country and Regions  Identified: Country and Subregions	The type of geographic region supported for conformant certificates.
<i>Maximum Certificate Chain Length</i>	Integer	Any value greater than 1, or “unbounded”	The maximum length from authorization certificate to root certificate of any certificate chain used by a SDEE. A received signed SPDU whose certificate chain is longer than this may be rejected.  SDEEs may have a maximum certificate chain length, but may also give guidance to developers that an appropriate certificate chain length is less than this maximum. For example, since long certificate chains increase packet size and therefore channel congestion and error rates, it is appropriate for the specification of the SDEE to give guidance that short (relative to the maximum) certificate chains should be used. This is particularly important for SDEEs that transmit frequently.
<i>Use Individual Linkage ID</i>	Boolean	True False	Whether to support individual linkage ID-based revocation in the certificates.
<i>Use Group Linkage ID</i>	Boolean	True False	Whether to support group linkage ID-based revocation in the certificates.
<i>Signature Algorithms in Chain or CRL</i>	Sequence of Enumerated	One or more of: ecdsaNistP256withSha1 ecdsaBrainpoolP256r1withSha1	The signature algorithms that may be used in the certificate chain or to sign CRLs relevant to the application.

### C.2.2 Maintenance of security profile policy

A number of parameters in the security profile could be the subject of a *security profile policy* for that SDEE, which in this context means that:

- a) There is an interest in consistent behavior across SDEE instances, so all SDEE instances at a given time and in a particular region should have the same values.
- b) There is a possibility that the appropriate value is going to change over time, for example as the number of participants in that SDEE increases.

An SDEE specification should note whether any of these parameters may need to be globally specified and updated, i.e., whether they are the subject of a changeable security profile policy. When SDEE instances based on that SDEE specification are deployed there should be mechanisms to update parameters that are governed by policy.

The following parameters may be particularly suitable to be subject to policy, as they can be changed without fundamentally changing the behavior of the invoking SDEE: *Data Validity Period, Data Validity Distance, Acceptable Future Data Period, Overdue CRL Tolerance, Signature Algorithm, Signature Algorithms in Chain or CRL*.

## C.3 IEEE 1609.2 security profile proforma<sup>13</sup>

### C.3.1 Instructions for completing the IEEE 1609.2 security profile proforma

The developer of an IEEE 1609.2 security profile may specify the profile by completing this proforma. The main part of the proforma is a fixed questionnaire, divided into entries. Answers to the questionnaire items are to be provided in the center column, and any elaboration necessary is to be provided in the rightmost column. The entries in the value column are either drawn from the list of permitted values given above, or are “n/a”.

---

<sup>13</sup> Copyright release for 1609.2 security profile proformas: Users of this standard may freely reproduce the 1609.2 security profile proforma in this annex so that it can be used for its intended purpose and may further publish the completed 1609.2 security profile.

### C.3.2 IEEE 1609.2 security profile proforma

#### C.3.2.1 IEEE 1609.2 security profile identification

Field	Value	Notes
<i>Name</i>		
<i>PSIDs</i>		
<i>Other considerations</i>		

#### C.3.2.2 Sending

Field	Value	Notes
<i>Sign Data</i>		
<i>Signed Data in Payload</i>		
<i>External Data</i>		
<i>External Data Source</i>		
<i>External Data Hash Algorithm</i>		
<i>Set Generation Time in Security Headers</i>		
<i>Set Generation Location in Security Headers</i>		
<i>Set Expiry Time in Security Headers</i>		
<i>Signed SPDU Lifetime</i>		
<i>Signer Identifier Policy Type</i>		
<i>Simple Signer Identifier Policy: Minimum Inter Cert Time</i>		
<i>Simple Signer Identifier Policy: Exceptions</i>		
<i>Simple Signer Identifier Policy: Signer Identifier Cert Chain Length</i>		
<i>Text Signer Identifier Policy</i>		
<i>Sign With Fast Verification</i>		
<i>EC Point Format</i>		
<i>p2pcd_useInteractiveForm</i>		
<i>p2pcd_maxResponseBackoff</i>		
<i>p2pcd_responseActiveTimeout</i>		
<i>p2pcd_requestActiveTimeout</i>		
<i>p2pcd_observedRequestTimeout</i>		
<i>p2pcd_currentlyUsedTriggerCertificateTime</i>		
<i>p2pcd_responseCountThreshold</i>		
<i>Repeat Signed SPDUs</i>		
<i>Time Between Signing</i>		
<i>Encrypt Data</i>		

### C.3.2.3 Receiving

Field	Value	Notes
<i>Use Preprocessing</i>		
<i>Verify Data</i>		
<i>Maximum Certificate Chain Length</i>		
<i>Relevance: Replay</i>		
<i>Relevance: Generation Time in Past</i>		
<i>Validity Period</i>		
<i>Relevance: Generation Time in Future</i>		
<i>Acceptable Future Data Period</i>		
<i>Generation Time Source</i>		
<i>Relevance: Expiry Time</i>		
<i>Expiry Time Source</i>		
<i>Consistency: Generation Location</i>		
<i>Relevance: Generation Location Distance</i>		
<i>Validity Distance</i>		
<i>Generation Location Source</i>		
<i>Overdue CRL Tolerance</i>		
<i>Relevance: Certificate Expiry</i>		
<i>Encrypted Data</i>		

### C.3.2.4 Security management

Field	Value	Notes
<i>Signing Key Algorithm</i>		
<i>Encryption Algorithm</i>		
<i>Implicit or Explicit Certificates</i>		
<i>EC Point Format</i>		
<i>Supported Geographic Regions</i>		
<i>Maximum Certificate Chain Length</i>		
<i>Use Individual Linkage ID</i>		
<i>Use Group Linkage ID</i>		
<i>Signature Algorithms in Chain or CRL</i>		

### C.3.2.5 Other

Field	Value	Notes
<i>Fields that may be subject to policy update</i>		

## C.4 Service Specific Permissions (SSP)

### C.4.1 General

As discussed in 5.2.3.3.3, the IEEE 1609.2 certificate provides two fields that are used to determine that the payload of a signed SPDU is consistent with the permissions of the sender. The PSID field indicates that the sender is entitled to send payloads associated with the application area indicated by the PSID field. The SSP field indicates that the sender has permissions to send specific payload types within that application area. The definition of application behavior within the application area includes the mapping from payload contents to the permissions (PSID and SSP) that determines the validity of the payload: in other words, one of the responsibilities of a PSID owner is to define the syntax and semantics of the SSP and to define which payloads are permitted by specific SSPs. The determination that a payload is consistent with the PSID and

SSP cannot be made by the SDS, as the SDS cannot know the full payload syntax associated with every PSID; this determination can only be implemented within the invoking SDEE.

As an example of SSP use, SSPs have been defined by the European Telecommunications Standards Institute (ETSI) for use by senders of the Cooperative Awareness Message (CAM) (ETSI EN 302 637-2 [B5]) and Decentralized Environmental Notification Message (DENM) (ETSI EN 302 637-3 [B6]). For CAM, the message contains several possible extension fields; the SSP defines which extension fields a sender can include, as well as optional fields within those extension fields.

As a further example of SSP use, this standard defines a SSP for CRLs in 7.4.3.

#### **C.4.2 SSP syntax and semantics**

A complete specification of a SDEE that uses the SSP includes a full definition of the syntax and semantics of the SSP and its relation to PDU payloads consumed by that SDEE, such that an unambiguous determination may be made as to whether or not a particular payload is permitted by a particular SSP. The organization defining the SDEE has responsibility to define the syntax and semantics of the SSP. The organization may follow the approach of ETSI in defining SSPs for CAM (ETSI EN 302 637-2 [B5]) and DENM (ETSI EN 302 637-3 [B6]), or of this standard in defining the SSP for CRLs in 7.4.3, or some other approach so long as it is unambiguous.

A PsidSsp structure as defined in this standard may omit the SSP. The definition of the SSP developed by the PSID owner should state whether this is permissible for a particular PSID, i.e., whether the PSID has a “default SSP”. If it is permissible for the SSP to be omitted, the definition of consistency between a payload and a SSP should include a definition of the meaning of the default PSID, i.e., a definition of the consistency conditions in the case where the SSP is omitted. There is no assumption in this standard about the meaning of an omitted PSID: it is simply a special case of SSP encoding. It would make sense, though, for the default SSP to be either the one that is going to be most frequently used or the one that corresponds to the minimal set of privileges for an entity entitled to use the PSID.

#### **C.5 Assurance level**

A complete specification of a SDEE includes an indication of whether the `assuranceLevel` field in the certificate (see 6.4.8) is used to validate SPDU contents, or to permit particular actions based on the SPDU contents.

#### **C.6 Recommendations on certificates**

The recommendations on certificates that should be noted in this section of the SDEE specification should be considered by certificate authorities (CAs) issuing certificates for that PSID.

- Whether there are restrictions on other SDEEs whose PSIDs may appear in the same certificate. For example, the certificates for a SDEE that naturally uses long-lived identities are not suitable for use by a SDEE that broadcasts frequently, as the long-lived identities of the first SDEE would allow the user to be tracked by observing broadcasts created by the second SDEE.

## Annex D

(informative)

### Examples and use cases

#### D.1 Guidance for SDEE specifiers and implementers

- a) A receiving secure data exchange entity (SDEE) should call Sec-SecureDataPreprocessing.request on every received secured protocol data unit (SPDU) to enable security management information to be correctly transferred: certificates are stored for later use, and peer-to-peer certificate distribution (P2PCD) is carried out as specified in Clause 8. Since the output of decryption and the contents of a signed SPDUs are both SPDUs themselves, a receiving SDEE should call Sec-SecureData-Preprocessing.request on those SPDUs as well.
- b) Some of the validity criteria in 5.2 are optional or may have SDEE-specific values associated with them, meaning that values given in the specification are used to develop the implementation of a particular SDEE. The specification of the receiving SDEE should indicate which optional criteria and which parameter values or ranges should be used. The specifier of a SDEE may use the *IEEE 1609.2 security profile* specified in Annex C as a compact way to specify which secure data service (SDS) parameters are used and which values they should take for that particular SDEE.
- c) For performance reasons, a receiving SDEE may not want to carry out all validity checks on a received SPDUs. This applies in particular to cryptographic validity checks, which are computationally expensive. A receiving SDEE should distinguish between SPDUs that have been fully validated and SPDUs that have not. The SDEE specification should indicate which actions are safe to take on SPDUs that have been validated and which actions, if any, the SDEE may take on SPDUs that have not been validated, or on which only some validation actions have been carried out. For example, a collision avoidance application may require that incoming SPDUs are signed and that the driver is only altered, or autonomous driving actions are only taken, on the basis of a signed SPDUs if that signature verifies.
- d) A receiving SDEE need not carry out all validity checks on a given SPDUs at the same time. Validity checks are a type of filter that allows protocol data units (PDUs) to be discarded, and there may be SDEE-specific filters as well. The SDEE implementer is free to choose which order to carry out validity checks and to mix SDEE-specific validity checks with the validity checks based on 5.2, so long as a clear distinction is maintained between SPDUs that have been fully validated according to the security profile and SPDUs that have not.
- e) Even if Sec-SignedDataVerification.request indicates that the signed SPDUs is valid, this does not demonstrate that the SPDUs meets all the validity conditions necessary to accept the message. There may be additional SDEE-specific validity conditions, as discussed in 5.2.3.3 and 5.2.4.3. A SDEE specification should specify what these conditions are, and an implementation of a SDEE should check that these conditions are satisfied. This standard identifies the following SDEE-specific validity conditions:
  - 1) Payload consistent with the permissions: the SDS cannot carry out these checks as it cannot parse the payload of the SPDUs.
    - i) **Provider service identifier (PSID).** A receiving SDEE should determine that the payload of the signed SPDUs is consistent with the PSID in the security envelope; this should be trivially true given the consistency requirement of 5.2.3.3.2.
    - ii) **Service Specific Permissions (SSP).** A design for a receiving SDEE may also place additional constraints on a signed SPDUs payload for that SDEE, using the SSP field in the sender's certificate to indicate those additional constraints. If SSP is present, the

receiving SDEE should check that the SPDU payload is consistent with the SSP according to the processing rules specified in the SDEE design. See 5.2.3.3.3 for further discussion.

- iii) **Assurance level.** A design for a receiving SDEE may also specify a minimum assurance level for particular signed SPDU payloads, based on the `assuranceLevel` field in the certificate. In this case, the receiving SDEE should check that the `assuranceLevel` in the certificate is appropriate for the received payload. See 5.2.3.3.3 for further discussion.
- 2) External data: The signature on a signed SPDU may be calculated over data that is not directly included in the payload of the signed SPDU. In this case, the hash of the external data is included. The SDEE specification should specify how this data is obtained by both sending and receiving SDEEs. The receiving SDEE should check that the external data hashes to the correct value. See 5.2.3.3.4.

## D.2 Processing CRLs

A certificate revocation list (CRL) verification process invokes `Sec-SignedDataVerification.request` to verify the CRL.

To check that the SSP is consistent with the CRL as specified in 7.3.3, the process:

- a) Checks that the CRL signer is consistent with the `associatedCraca` field in the SSP.
- b) Checks that the CRL series in the CRL payload is contained in the `crls` field in the SSP.

To check that the CRL signer is consistent with the `associatedCraca` field in the SSP, the process does the following:

- a) If `associatedCraca` field in the SSP is equal to `isCraca`, the process determines that the Certificate Revocation Authorizing Certificate Authority (CRACA) certificate is the certificate that signed the CRL. To do this:
  - 1) The process extracts the `SignerIdentifier` from the `SignedData` containing the signed CRL.
  - 2) If the `SignerIdentifier` is of type `digest`, and if the `digest` field is equal to the `cracaId` field in the CRL payload, the consistency check succeeds.
  - 3) If the `SignerIdentifier` is of type `certificate`, and if the `HashedId8` of the certificate is equal to the `cracaId` field in the CRL payload, the consistency check succeeds.
  - 4) Otherwise, the consistency check fails.
- b) If `associatedCraca` field in the SSP is equal to `issuerIsCraca`, the process determines that the CRACA certificate is the certificate that issued the certificate that signed the CRL. To do this:
  - 1) The process obtains the certificate `crlSignCert` that signed the CRL:
    - i) The process extracts the `SignerIdentifier` from the `SignedData` containing the signed CRL.
    - ii) If the `SignerIdentifier` is of type `digest`, the process invokes `SSME-CertificateInfo.request` with parameters `Identifier Type` to obtain `crlSignCert`.

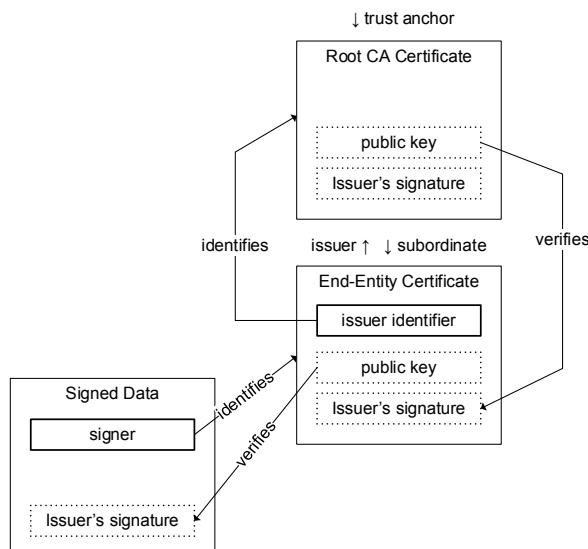
- iii) If the SignerIdentifier is of type `certificate`, then `crlSignCert` is the first certificate in the provided array as specified in 6.3.24.
- 2) The consistency check succeeds if:
    - i) The field `crlSignCert.issuer` is of type `sha256AndDigest`.
    - ii) The field `crlSignCert.issuer/sha256AndDigest` is equal to the `cracaid` field in the CRL payload.
  - 3) Otherwise, the consistency check fails.

If the CRL is valid, the CRL receiving process extracts the revocation information about the individual certificates and stores it in the security services management entity (SSME) via `SSME-AddHashIdBasedRevocation.request`, `SSME-AddIndividualLinkageBasedRevocation.request`, or `SSME-AddGroupLinkageBasedRevocation.request`. It provides additional information about the CRL, such as the next CRL issue date, via `SSME-AddRevocationInfo.request`.

### D.3 Constructing a certificate chain

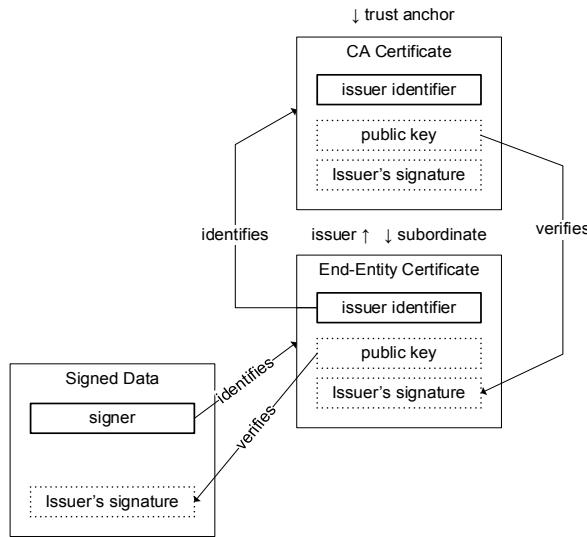
#### D.3.1 Examples

Figure D.1 shows a simple certificate chain of length 2 (the length of a certificate chain is the number of certificates in it). The trust anchor is the root certificate authority (CA) certificate. The authorization certificate and the root CA certificate are both explicit certificates. The signer id in the authorization certificate identifies the root CA certificate that issued it. The public key in the root CA certificate is used to verify the signature on the authorization certificate. If the authorization certificate is used to sign a SPDU, the signer field in the signed SPDU identifies the specific authorization certificate that signed the SPDU and the public key field in the authorization certificate is used to verify the signature on the SPDU.



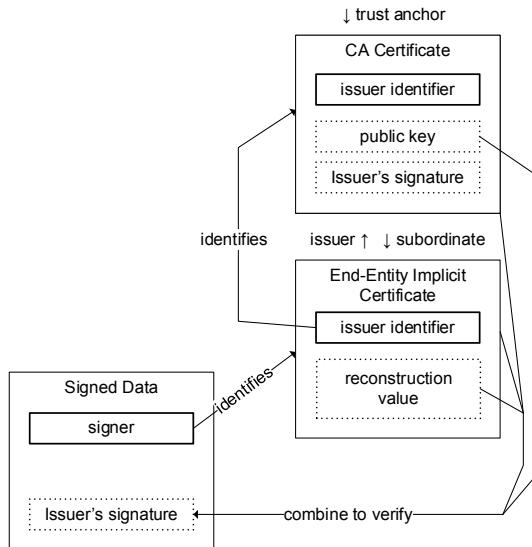
**Figure D.1—Length-2 certificate chain with explicit certificates and a root CA as trust anchor**

Figure D.2 shows a certificate chain of length 2 where the authorization certificate and the trust anchor are both explicit certificates. In this case the trust anchor is a CA certificate rather than a root certificate. Other than that the chain is identical to the one presented in Figure D.1.



**Figure D.2—Length-2 certificate chain with explicit certificates and a non-root CA as trust anchor**

Figure D.3 shows a certificate chain of length 2 where the authorization certificate is an implicit certificate. As required, the trust anchor is an explicit certificate. As with explicit certificates, the signer id in the authorization certificate identifies the root CA certificate that issued it. However, in this case the issuing certificate does not sign the subordinate certificate. Instead, to cryptographically verify the certificate, an operation is performed combining the hash of the issuing certificate, the hash of the authorization certificate, the reconstruction value from the authorization certificate, and the public key from the CA certificate to reconstruct the authorization's public key as specified in 5.3.2. If that public key cryptographically verifies the signature, this provides assurance both that the authorization did, in fact, sign the SPDU and that the certificate was validly issued by the CA.

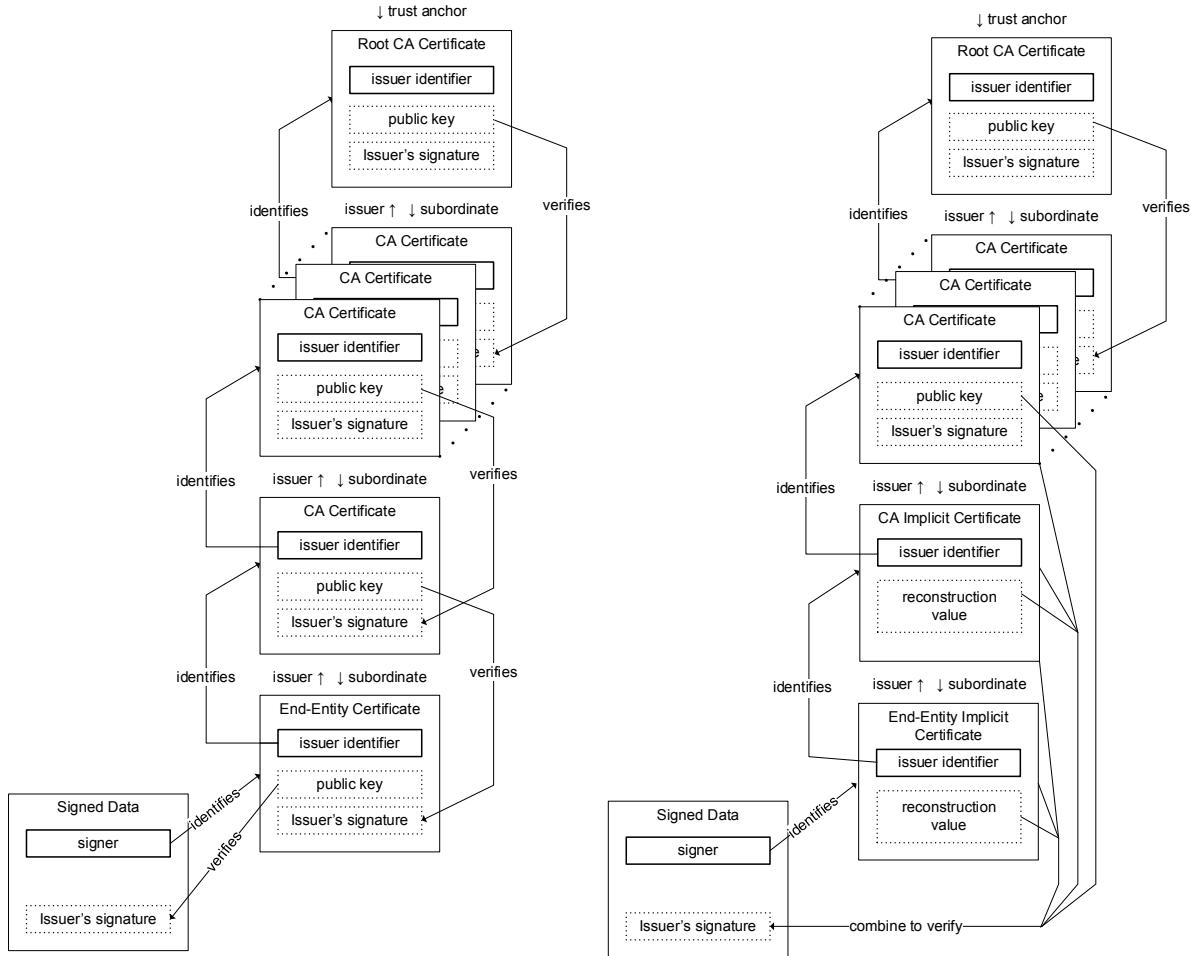


**Figure D.3—Length-2 certificate chain with implicit certificates and a non-root CA as trust anchor**

Figure D.4 illustrates two longer certificate chains. It shows that for each pair of certificates in the chain, the subordinate certificate contains an issuer identifier identifying the issuing certificate and the issuing certificate's public key verifies the subordinate certificate either explicitly or implicitly.

The chain on the left contains explicit certificates only. The public key in each issuing certificate verifies the signature of its subordinate certificate.

The chain on the right ends with two implicit certificates. The validity of the implicit certificates is demonstrated by combining the hashes of all the implicit certificates, the reconstruction values of all the implicit certificates, and the hash and public key from the first explicit certificate in the chain to verify the signature on the signed data as specified in 5.3.2.



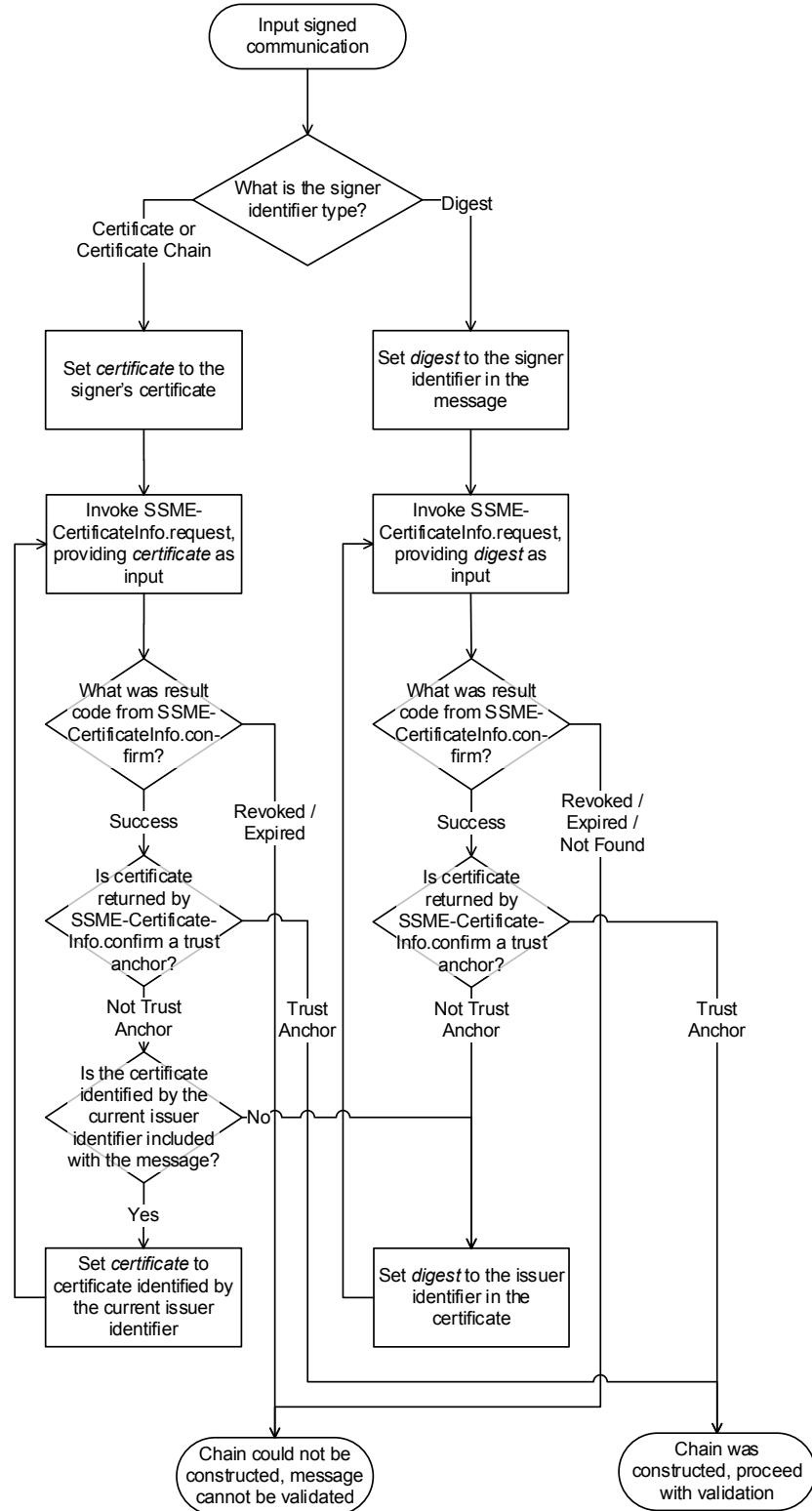
**Figure D.4—Long certificate chains, one with all explicit certificates (left) and one ending with two implicit certificates (right)**

### D.3.2 Construction

A signed SPDUs need not include all the certificates in the chain. Instead, the SPDUs may include a chain of one or more certificates that does not reach all the way back to a trust anchor or the SPDUs may omit all certificates and instead include a reference to its signing certificate using a hash. In this case it may be possible to construct the chain using certificates that are stored by the SSME. If the SDS encounters a certificate hash for which the corresponding certificate is not included in the PDU, the services may invoke the SSME-CertificateInfo.request primitive to determine whether that hash corresponds to a certificate that is already known to the SSME. Figure D.5 illustrates the logic flow to be used when constructing a certificate chain. The processing cycles through the certificates that were received with the signed SPDUs, and when no received certificate matches the current signer identifier, the processing invokes SSME-CertificateInfo.request to attempt to continue building it.

It is conceivable that the issuer identifier in a certificate may identify two different certificates known to the SSME. This will happen if the HashedId8 of the two certificates, i.e., the low-order 8 bytes of the SHA-256 hashes of the two certificates, are identical. For any pair of certificates, the probability that the two HashedId8 values collide is  $2^{-64}$ . If the SSME stores  $k$  certificates, the probability that there is at least one collision is roughly equal to  $k^2 \times 2^{-64}$ . In the unlikely event of a collision, the SDS builds both certificate chains. If one does not have consistent permissions it is discarded. If both have consistent permissions they

are both cryptographically verified. In this case if either chain verifies correctly, that chain is taken to be valid and is able to validate the signed SPDU.



**Figure D.5—Logic flow for constructing a certificate chain**

## D.4 Peer-to-peer certificate distribution

### D.4.1 General

D.4 gives an example of how P2PCD can be implemented using the primitives defined in this standard along with internal state variables.

### D.4.2 State, timers, and configuration parameters within SSME

#### D.4.2.1 State within SSME

##### D.4.2.1.1 Request

In this example, the SSME maintains the following state variables to support P2PCD request:

*isRequestActive* (p2pcdLearningRequest  $c$ , SDEE ID  $s$ ):

- **Type:** Boolean.
- **Meaning:** True if the SSME has seen an incoming or created an outgoing learning request for  $c$  associated with SDEE  $s$  within a configurable period, such that it does not create another learning request for  $c$ . False otherwise.
- **Change conditions:** Set to “False” on initialization. Set to “True” by SSME-Sec-IncomingP2pcd-Info.request or SSME-Sec-OutgoingP2pcdInfo.request. Set to “False” on the expiry of the timer *p2pcdRequestBackoffTimer* (p2pcdLearningRequest  $c$ , SDEE ID  $s$ ).

*queuedMissingCertIndicators* (SDEE ID  $s$ ):

- **Type:** array of HashedId8  $h$ .
- **Meaning:** The array of certificate identifiers for certificates which the SSME does not know and for which there is not an outstanding learning request. The SSME selects from this array when generating a p2pcdLearningRequest as specified in D.4.3.1 step d)1).
- **Change conditions:** Initialized to being empty. Entries are added to this array by SSME-Sec-IncomingP2pcdInfo.request when SSME-Sec-IncomingP2pcdInfo.request passes a certificate with an unknown certificate on its chain. Entries are removed from this array by:
  - SSME-AddCertificate.request, when the certificate passed via the primitive indicates a HashedId8 in the array.
  - SSME-Sec-OutgoingP2pcdInfo.confirm, when a p2pcdLearningRequest for that entry is included in a signed SPDU.
  - SSME-Sec-IncomingP2pcdInfo.request, when the p2pcdLearningRequest passed via the primitive indicates a certificate in the array.
  - Optionally, after a timeout period if they have not been used to form a p2pcdLearningRequest field.

#### D.4.2.1.2 Response

The SSME maintains the following state variables to support P2PCD response:

*isResponseActive* (p2pcdLearningRequest  $c$ , SDEE ID  $s$ ):

- **Type:** Boolean.
- **Meaning:** True if the SSME is in in the time-out period during which it responds no more than once to a request  $c$  for SDEE  $s$ . False otherwise.
- **Change conditions:** Set to “False” on initialization. Set to “True” by SSME-Sec-IncomingP2pcd-Info.request. Set to “False” on the expiry of the timer  $p2pcdResponseActiveTimer$  (p2pcdLearningRequest  $c$ , SDEE ID  $s$ ).

*p2pcdResponseCount* (p2pcdLearningRequest  $c$ , SDEE ID  $s$ ):

- **Type:** Integer.
- **Meaning:** Indicates the number of responses observed to  $c$  since the start of the current response-active period.
- **Change conditions:** Initialized to 0 by SSME-Sec-IncomingP2pcdInfo.request if a request is received for a certificate that was recently used by SDEE  $s$  and if the SSME is not already considering a response to an identical request  $c$ . Incremented by SSME-AddCertificate.request when the certificate added by SSME-AddCertificate.request is the one indicated by  $c$ .

*recentlyUsedSigningCertificates* (SDEE ID  $s$ ):

- **Type:** tuple of (CA certificate  $c$ , time added  $t$ ).
- **Meaning:** The array of certificates which are “recently used” by the SSME, i.e., the certificates for which the SSME sends a response if (a) they are requested and (b) the threshold condition is met.
- **Change conditions:** Entries are added to this array via SSME-Sec-OutgoingP2pcdInfo.request. Entries are removed from this array once their value of  $t$  is in the past by more than  $p2pcd\_currentlyUsedTriggerCertificateTime$ .

#### D.4.2.2 Timers within SSME

In this example, the SSME uses timers to support P2PCD. A timer is a block of functionality supporting the following functions:

- a) Initialize: The timer is initialized with a timeout interval.
- b) Re-initialize: The timeout interval for the timer is reset.
- c) Expire: The timer expires, potentially causing an action to be taken by the SSME.

The SSME uses the following timers:

*p2pcdRequestActiveTimer* (p2pcdLearningRequest  $c$ , SDEE ID  $s$ ):

- **Meaning:** Prevents the SSME from sending two requests within the same time interval, or from sending a request if it knows a request to be active.
- **Initialized by:** SSME-Sec-IncomingP2pcdInfo.request.

- **Re-initialized by:** SSME-Sec-IncomingP2pcdInfo.request, SSME-Sec-OutgoingP2pcdInfo.request.
- **On expiry:** SSME sets *isResponseActive(c, s)* to “False”, allowing it to start a fresh response cycle.

*p2pcdResponseBackoffTimer* (p2pcdLearningRequest *c*, SDEE ID *s*):

- **Meaning:** Used to determine when to decide whether or not to request that a P2PCD response is sent.
- **Initialized by:** SSME-Sec-IncomingP2pcdInfo.request.
- **Re-initialized by:** None.
- **On expiry:** SSME determines whether to send a response as specified in D.4.3.5.

*p2pcdResponseActiveTimer* (p2pcdLearningRequest *c*, SDEE ID *s*):

- **Meaning:** Prevents the SSME from sending two responses within the same time interval.
- **Initialized by:** SSME-Sec-IncomingP2pcdInfo.request.
- **Re-initialized by:** None.
- **On expiry:** SSME sets *isResponseActive(c, s)* to “False”, allowing it to start a fresh response cycle.

### D.4.3 Activities within P2PCD

#### D.4.3.1 General

Subclause D.4.3 describes processing for the following events:

- a) Receive trigger SDEE SPDUs (see D.4.3.2).
- b) Send trigger SDEE SPDUs (see D.4.3.3).
- c) Register for response generation service (see D.4.3.4).
- d) Send P2PCD learning response (see D.4.3.5).
- e) Receive P2PCD learning response (see D.4.3.6).
- f) p2pcdRequestActiveTimer or p2pcdResponseActiveTimer expire (see D.4.3.7).

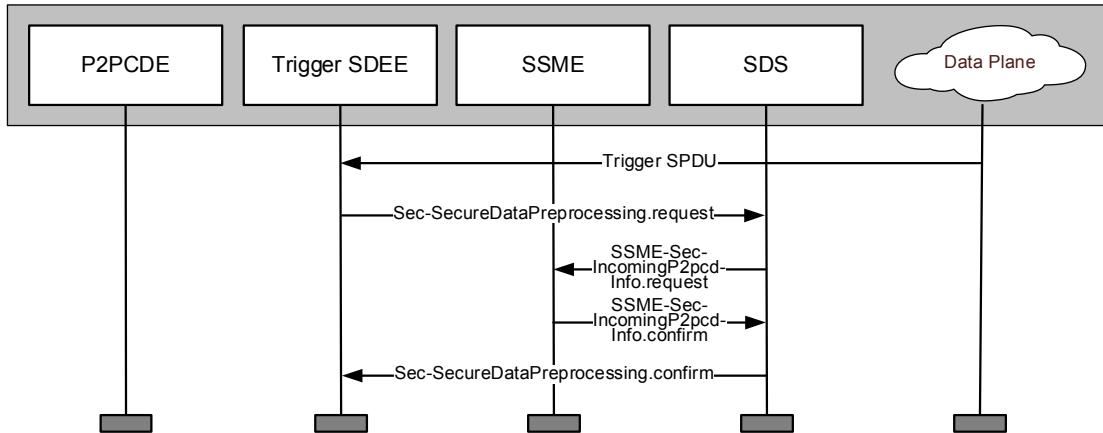
#### D.4.3.2 Receive trigger SDEE SPDUs

Receiving trigger SDEE SPDUs proceeds as follows. The flow is illustrated in Figure D.6.

- a) The *trigger SDEE* receives a SPDU.
- b) The trigger SDEE invokes the SDS via Sec-SecureDataPreprocessing.request, passing the SPDU and its SDEE ID.
- c) If the SPDU is of type signed:
  - 1) If the SignerIdentifier in the signed SPDU indicates the selection certificate, and/or if the HeaderInfo in the SPDU contains a p2pcdLearningRequest field, the SDS invokes SSME-Sec-IncomingP2pcdInfo.request with the parameters:
    - i) *SDEE ID:* The SDEE ID of the trigger SDEE.

- ii) *Certificate*: the certificates contained in the *certificate* field from the *SignerIdentifier* within the SPDU (if present). Recall that this field may contain one or more certificates.
  - iii) *P2pcdLearningRequest*: the *p2pcdLearningRequest* value from the SPDU (if present).
- d) **Requester role SSME processing:**
- 1) If the *Certificate* parameter was provided, then the SSME determines whether the certificates reference an unknown certificate as follows:
    - i) If the *issuer* field in any *Certificate* within the parameter *Certificate* indicates a certificate, *Issuer*, that is not known to the SSME as defined in 4.3, then:
      - i) The SSME calculates  $c$ , the *HashedId3* of *Issuer*.
      - ii) If  $\text{isRequestActive}(c, \text{SDEE ID})$  is False, then
        - i) The SSME adds the *HashedId8* of *Issuer* to *queuedMissingCertIndicators* (*SDEE ID*).
  - 2) If the *P2pcdLearningRequest* parameter was provided, then the SSME determines whether this is a request for a certificate that the SSME is currently requesting, and if so extends the timeout for sending a second request, as follows:
    - i) If the trigger certificate identified in *P2pcdLearningRequest* is not known to the SSME as defined in 4.3, then:
      - i) If the timer *p2pRequestActiveTimer* (*P2pcdLearningRequest*, *SDEE ID*) is not initialized, or is initialized but will expire in less time than *p2pcd\_observedRequestTimeout*(*SDEE ID*), the SSME initializes (or re-initializes) that the timer with timeout value *p2pcd\_observedRequestTimeout*( $s$ ).
      - ii) If *isRequestActive* is False, then the SSME sets *isRequestActive* (*P2pcdLearningRequest*, *SDEE ID*) to True.
      - iii) If *P2pcdLearningRequest* corresponds to one of the entries in *queuedMissingCertIndicators* (*SDEE ID*) as defined in 8.4.2, then the SSME removes that entry from *queuedMissingCertIndicators* (*SDEE ID*).
- e) **Responder role SSME processing:** If *P2pcdLearningRequest* parameter was provided, then the SSME determines whether it should consider responding to the request as follows:
- 1) If *P2pcdLearningRequest* corresponds to a CA certificate stored in *recentlyUsedSigningCertificates*(*SDEE ID*) (where “corresponds to” is defined in 8.4.2) and if the time  $t$  associated with that certificate in *recentlyUsedSigningCertificates*(*SDEE ID*) is within *p2pcd\_currentlyUsedTriggerCertificateTime* (*SDEE ID*) of the current time, then:
    - i) If *isResponseActive* (*p2pcdLearningRequest*, *SDEE ID*) is “False”, then
      - i) The SSME sets *isResponseActive* (*P2pcdLearningRequest*, *SDEE ID*) to “True”.
      - ii) The SSME sets *p2pcdResponseCount* (*P2pcdLearningRequest*, *SDEE ID*) to “0”.
      - iii) The SSME initializes the timer *p2pcdResponseBackoffTimer* with expiry time chosen randomly between 0 and *p2pcd\_maxResponseBackoff*(*SDEE ID s*).
      - iv) The SSME initializes the timer *p2pcdResponseActiveTimer* with expiry time equal to *p2pcd\_responseActiveTimeout*(*SDEE ID s*).
      - v) Go to step f).

- ii) If *isResponseActive* (*P2pcdLearningRequest*, *SDEE ID*) is “True”, go to step f).
- 2) Otherwise, go to step f).
- f) The SSME confirms to the SDS that this has been carried out via SSME-Sec-IncomingP2pcd-Info.confirm.
- g) The SDS confirms to the trigger SDEE that this has been carried out via Sec-SecureDataPre-processing.confirm.



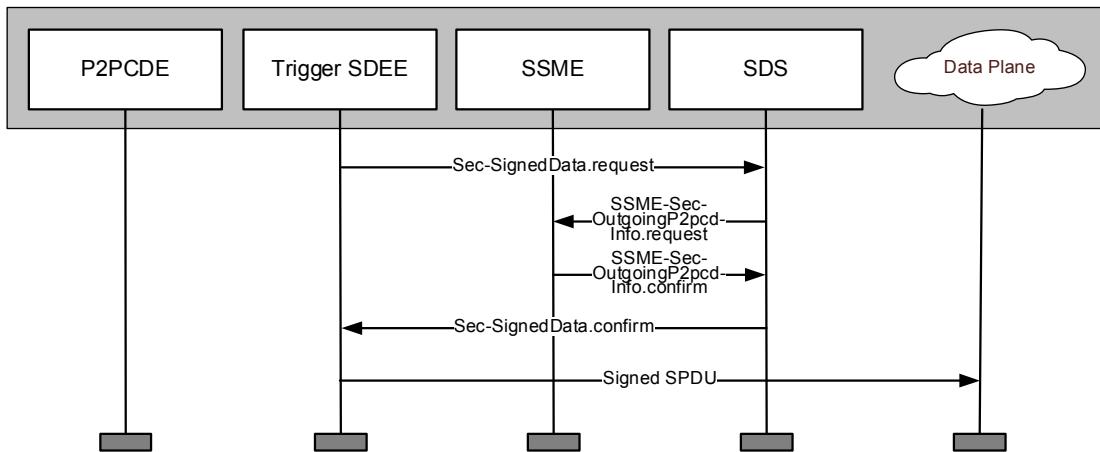
**Figure D.6—P2PCD operations: receiving SPDUs for trigger SDEE**

#### D.4.3.3 Send trigger SDEE SPDUs

Sending trigger SDEE SPDUs proceeds as follows. The flow is illustrated in Figure D.7.

- a) The trigger SDEE invokes the SDS via Sec-SignedData.request, passing the SDEE ID.
- b) The SDS invokes SSME-Sec-OutgoingP2pcdInfo.request, passing it the SDEE ID and the signing certificate.
- c) **Responder role SSME processing:**
  - 1) The SSME constructs or looks up the signing certificate chain. For each CA certificate  $c$  in the chain, the SSME uses  $(c, \text{current time})$  to update the array *recentlyUsedSigningCertificates*(*SDEE ID*), either by adding  $c$  if it is not present or by updating the time associated with  $c$  if it is present.
- d) **Requester role SSME processing:**
  - 1) If *queuedMissingCertIndicators* (*SDEE ID*) is not empty, then the SSME creates a *p2pcdLearningRequest*:
    - i) The SSME selects one of the entries  $h$  in *queuedMissingCertIndicators* (*SDEE ID*). This is a HashedId8.
    - ii) The SSME calculates  $c$ , a HashedId3, which is the P2PCD learning request value associated with  $h$ . This is calculated as specified in 8.4.2.
    - iii) The SSME removes  $h$  from *queuedMissingCertIndicators* (*SDEE ID*).
    - iv) The SSME sets *isRequestActive*( $c$ , *SDEE ID*) to True.
    - v) The SSME initializes the timer *p2pRequestActiveTimer*( $c$ , *SDEE ID*).

- 2) The SSME returns  $c$  as the *p2pcdLearningRequest* parameter to SSME-Sec-OutgoingP2pcd-Info.confirm, or omits that parameter.
- e) The SDS includes *p2pcdLearningRequest*, if it was provided, when creating the HeaderInfo for the ToBeSignedData prior to signing, per requirement a).
- f) The SDS returns the signed SPDU, including the *p2pcdLearningRequest* if appropriate, to the trigger SDEE via Sec-SignedData.confirm.

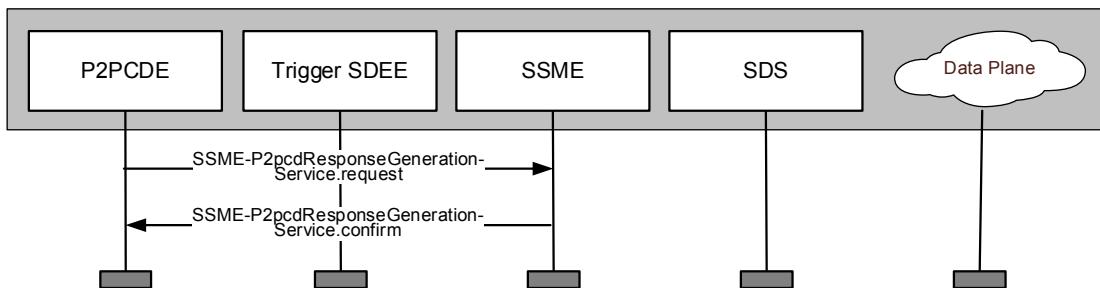


**Figure D.7—P2PCD operations: sending SPDU for trigger SDEE**

#### D.4.3.4 Register for response generation service

A P2PCD Entity registers for the P2PCD response generation service as follows. The flow is illustrated in Figure D.8.

- a) The Peer-to-peer Certificate Distribution application (P2PCDE in the figure) uses SSME-P2pcdResponseGenerationService.request to request that the SSME notifies it when P2PCD learning responses should be sent for requests related to a particular SDEE.
- b) The SSME confirms the request via SSME-P2pcdResponseGenerationService.confirm.

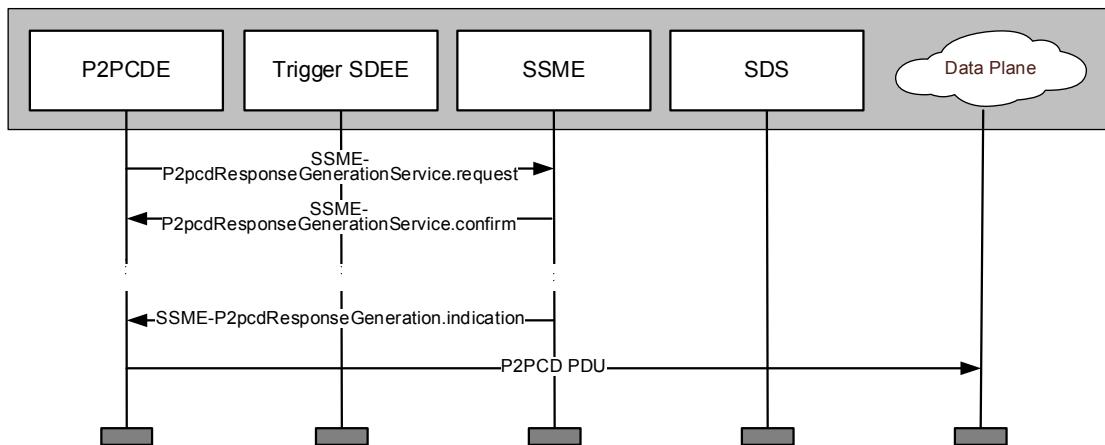


**Figure D.8—P2PCD operations: P2PCD Entity registers for response generation service**

#### D.4.3.5 Send P2PCD learning response

Sending P2PCD learning responses proceeds as follows. The flow is illustrated in Figure D.9.

- a) When the timer  $p2pcdResponseBackoffTimer$  ( $p2pcdLearningRequest c$ , SDEE ID  $s$ ) for any  $(c, s)$  expires:
  - 1) If  $p2pcdResponseCount (c, s)$  is less than or equal to  $p2pcd\_responseCountThreshold (s)$ , then:
    - i) The SSME creates an array of certificates such that the first certificate in the array is the requested CA certificate, the last certificate in the array was issued by a root certificate, and each certificate in the array other than the first is the issuer of the one before it.
    - ii) The SSME generates a notification to the P2PCD application via `SSME-P2pcdResponseGeneration.indication` indicating that the P2PCD application should create and send a P2PCD learning response containing the indicated certificates.
  - 2) The P2PCD application creates a P2PCD application PDU per 8.4 and sends it over the data plane.



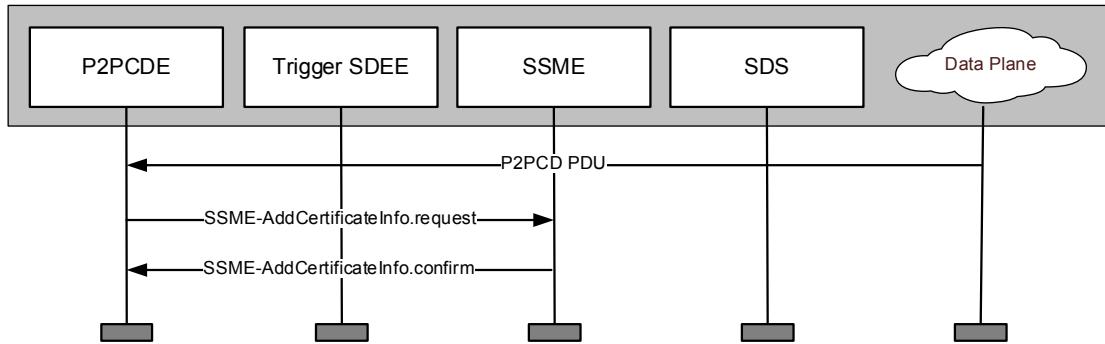
**Figure D.9—P2PCD operations: P2PCD Entity sends response**

#### D.4.3.6 Receive P2PCD learning response

Receiving P2PCD learning responses proceeds as follows. The flow is illustrated in Figure D.10. In order to receive responses, the P2PCDE registers with the network stack to receive incoming data on the appropriate TCP/IP port or WSMP PSID.

- a) The P2PCDE receives a P2PCD application PDU over the data plane containing CA certificates.
- b) The P2PCD provides the certificates to the SSME via `SSME-AddCertificate.request`, optionally verifying them beforehand via `SSME-VerifyCertificate.request`.
- c) For each certificate provided to the SSME:
  - 1) The SSME calculates the corresponding `HashedId8` value  $h8$  and `P2pcdLearningRequest` value  $clr$  as specified in 8.4.2.

- 2) The SSME removes  $h8$  from any  $queuedMissingCertIndicators(s)$  array in which it appears, per requirement d).
- 3) The SSME increments  $p2pcdResponseCount(c, s)$  by one for any instance of  $p2pcdResponseCount(c, s)$  for which  $c = clr$ .
- 4) The SSME sets  $isRequestActive(c, s)$  to False for any instance of  $isRequestActive(c, s)$  for which  $c = clr$ , per requirement d).
- 5) The SSME confirms the operation via SSME-AddCertificate.confirm.



**Figure D.10—P2PCD operations: P2PCD Entity receives response**

#### D.4.3.7 p2pcdRequestActiveTimer or p2pcdResponseActiveTimer expire

The expiry of the timers  $p2pcdRequestActiveTimer$  and  $p2pcdResponseActiveTimer$  is handled internally to the SSME and does not involve communication across an interface.

When  $p2pcdRequestActiveTimer(c, s)$  expires, the SSME sets  $isRequestActive(c, s)$  to “False”.

When  $p2pcdResponseActiveTimer(c, s)$  expires, the SSME sets  $isResponseActive(c, s)$  to “False”.

### D.5 Example data structures

#### D.5.1 “Basic safety message” with dummy payload, signed with a digest

##### D.5.1.1 Description

This is an example of a SPDPU, which is an Ieee1609Dot2Data, that uses the security profile for Basic Safety Message provided in SAE J2945/1 [B21] and the PSID for “vehicle to vehicle safety and awareness” specified in IEEE Std 1609.12. The payload is not a valid BSM but the ASCII string “This is a BSM\r\n”. The SignerIdentifier field is of type *digest*. The payload is 15 bytes long and the entire SPDPU is 108 bytes long.

##### D.5.1.2 COER encoding

03	81	00	40	03	80	0F	54	68	69	73	20	69	73	20	61
20	42	53	4D	0D	0A	40	01	20	11	12	13	14	15	16	17

```

18 80 21 22 23 24 25 26 27 28 80 82 31 32 33 34
35 36 37 38 31 32 33 34 35 36 37 38 31 32 33 34
35 36 37 38 31 32 33 34 35 36 37 38 41 42 43 44
45 46 47 48 41 42 43 44 45 46 47 48 41 42 43 44
45 46 47 48 41 42 43 44 45 46 47 48

```

#### **D.5.1.3 ASN.1 value notation**

```

value1 Ieee1609Dot2Data ::= {
    protocolVersion 3,
    content signedData : {
        hashId sha256,
        tbsData {
            payload {
                data {
                    protocolVersion 3,
                    content unsecuredData : '5468697320697320612042534D0D0A'H
                }
            },
            headerInfo {
                psid 32,
                generationTime 1230066625199609624 -- hex: 1112131415161718
            }
        },
        signer digest : '2122232425262728'H,
        signature ecdsaNistP256Signature : {
            r compressed-y-0 :
'3132333435363738313233343536373831323334353637383132333435363738'H,
            s
'4142434445464748414243444546474841424344454647484142434445464748'H
        }
    }
}

```

#### **D.5.2 “Basic safety message” with dummy payload, signed with a certificate**

##### **D.5.2.1 Description**

This is an example of a SPDU, which is an Ieee1609Dot2Data, that uses the security profile for Basic Safety Message provided in SAE J2945/1 [B21] and the PSID for “vehicle to vehicle safety and awareness” specified in IEEE Std 1609.12. The payload is not a valid BSM but the ASCII string “This is a BSM\r\n”. The SignerIdentifier field is of type *certificate* and contains a single certificate. The payload is 15 bytes long and the entire SPDU is 207 bytes long.

##### **D.5.2.2 COER encoding**

```

03 81 00 40 03 80 0F 54 68 69 73 20 69 73 20 61
20 42 53 4D 0D 0A 40 01 20 11 12 13 14 15 16 17
18 81 01 01 00 03 01 80 21 22 23 24 25 26 27 28
50 80 80 00 64 31 32 33 34 35 36 37 38 39 41 42
43 44 51 52 53 54 55 56 57 58 59 61 62 63 00 46
04 E0 9A 20 84 00 A9 83 01 03 80 00 7C 80 01 E4
80 03 48 01 02 00 01 20 00 01 26 81 82 91 92 93

```

```
94 95 96 97 98 91 92 93 94 95 96 97 98 91 92 93
94 95 96 97 98 91 92 93 94 95 96 97 98 80 82 31
32 33 34 35 36 37 38 31 32 33 34 35 36 37 38 31
32 33 34 35 36 37 38 31 32 33 34 35 36 37 38 41
42 43 44 45 46 47 48 41 42 43 44 45 46 47 48 41
42 43 44 45 46 47 48 41 42 43 44 45 46 47 48
```

### D.5.2.3 ASN.1 value notation

```
value1 Ieee1609Dot2Data ::= {
    protocolVersion 3,
    content signedData : {
        hashId sha256,
        tbsData {
            payload {
                data {
                    protocolVersion 3,
                    content unsecuredData : '5468697320697320612042534D0D0A'H
                }
            },
            headerInfo {
                psid 32,
                generationTime 1230066625199609624 -- hex: 1112131415161718
            }
        },
        signer certificate : {
            {
                version 3,
                type implicit,
                issuer sha256AndDigest : '2122232425262728'H,
                toBeSigned {
                    id linkageData : {
                        iCert 100,
                        linkage-value '313233343536373839'H,
                        group-linkage-value {
                            jValue '41424344'H,
                            value '515253545556575859'H
                        }
                    },
                    cracaId '616263'H,
                    crlSeries 70,
                    validityPeriod {
                        start 2172814212 -- hex: 81828384,
                        duration hours : 169
                    },
                    region identifiedRegion : {
                        countryOnly : 124,
                        countryOnly : 484,
                        countryOnly : 840
                    },
                    appPermissions {
                        {
                            psid 32
                        },
                        {
                            psid 38
                        }
                    }
                }
            }
        }
    }
}
```

```
        }
    },
    verifyKeyIndicator reconstructionValue : compressed-y-0 :
'9192939495969798919293949596979891929394959697989192939495969798'H
}
}
},
signature ecdsaNistP256Signature : {
    r compressed-y-0 :
'3132333435363738313233343536373831323334353637383132333435363738'H,
    s
'4142434445464748414243444546474841424344454647484142434445464748'H
}
}
}
```

## Annex E

(informative)

### Deployment considerations

The services specified in this standard do not provide a complete security system. The following aspects should also be considered when a system based on this standard is to be deployed.

- **Other communications models:** If two peer processes' security requirements are best met by secure sessions, rather than individual message security, what mechanism should be used to achieve this?
- **Privacy protection:** The IEEE 1609.2 design allows a device to protect privacy by changing its certificate. What other steps should a device take to protect privacy? For example, should it change other identifiers in the stack? Should it take steps to protect against an eavesdropper associating two different applications hosted on the same device? How often should it change certificates and other identifiers? The Preciosa Project has provided additional material for consideration on this topic [B19].
- **Root certificate authority (CA) certificate management:** Should the system support multiple root CA instances, or root CA certificate rollover? How is this implemented if so?
- **Certificate management:** How do devices that use IEEE 1609.2 certificates obtain them? See Whyte, et al. [B24] for a proposed design at the architecture rather than protocol specification level.
- **Peer-to-peer certificate distribution (P2PCD) over WAVE Short Message Protocol (WSMP):** Which channel is used for P2PCD learning responses? It makes sense for this to be in general the same channel as is used for secured protocol data units (SPDUs) from the trigger secure data exchange entity (SDEE).
- **Other scenarios for P2PCD:** The P2PCD mechanism specified in Clause 8 relies on both the requesting and the responding device being able to send SPDUs for the trigger SDEE. This will not always be the case—for example, cars can receive signed signal phase and timing messages from traffic signals but cannot send them. How does the system address these scenarios?
- **Cryptomaterial management:** How is cryptomaterial protected on the device? How do application instances that make use of cryptomaterial reference that cryptomaterial? How does the device prevent access to cryptomaterial by unauthorized processes? See the SeVeCom report [B22] for additional discussion on this and other security topics.
- **Certificate revocation list (CRL) distribution:** How are CRLs delivered to the CRL Verification Entity on a device?
- **CRL timing:** Should it be possible to issue CRLs before the nextCrl date in their predecessor? Are there special considerations redistributing these CRLs as opposed to “in-cycle” CRLs?
- **Revocation:** How is the decision made to revoke a certificate or set of certificates?
- **Misbehavior and intrusion detection:** Are incoming messages monitored to determine if they are potentially malicious? Is some networked service notified of potentially malicious messages? How should devices respond to a misbehaving CA, for example a CA which issues some certificates that are inconsistent with its certificate-issuance permissions?
- **Hardware security requirements:** What are the requirements for secure operation of a device hosting an instance of WAVE Security Services? What protection should it provide against physical attacks? Should it be certifiable in accordance with external standards for cryptographic modules such as FIPS 140-2 [B8]?

- **Random number generation:** How do devices obtain random numbers to avoid attacks on weak random number generators? See, for example, ANSI X9.82-1:2006 [B1], Goldberg and Wagner [B9], NIST (SP) 800-90A [B15], and Debian security advisory [B23]. In particular, Elliptic Curve Digital Signature Algorithm (ECDSA) is vulnerable to weak random number generators: see Howgrave-Graham and Smart [B10], and Nguyen and Shparlinski [B16]. A strong random number generator is necessary for both key and signature generation.
- **Local considerations:** What may WAVE Service Advertisements (WSAs) advertise? How are geographic subregions defined, for example are they defined by reference to ISO 3166-2 [B14]?

## Annex F

(informative)

### Bibliography

- [B1] ANSI X9.82-1:2006, Random Number Generation Part 1: Overview and Basic Principles, ANSI, 2006, available from <http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.82-1%3a2006>.
- [B2] Antipa, A., D. R. Brown, R. Gallant, R. Lambert, R. Struik, and S. A. Vanstone, “Accelerated verification of ECDSA signatures,” *Selected Areas in Cryptography, 12th International Workshop, SAC 2005*, Kingston, ON, Canada, pp. 307–318, Aug. 11–12, 2005.
- [B3] Brown, D., R. Gallant, and S. Vanstone, “Provably secure implicit certificate schemes,” *Financial Cryptography*, pp. 156–165, July 2002.
- [B4] Brown, D. R. L., M. J. Campagna, and S. A. Vanstone, “Security of ECQV-Certified ECDSA Against Passive Adversaries,” *Cryptology ePrint Archive, 2009-620*, Mar. 9, 2011. Available from <http://eprint.iacr.org/2009/620.pdf>.
- [B5] ETSI EN 302 637-2 V1.2.1 (2011-03), Technical Specification: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service.
- [B6] ETSI EN 302 637-3 V1.2.1 (2014-09), Technical Specification: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specification of Decentralized Environmental Notification Basic Service.
- [B7] ETSI TS 103 097, Intelligent Transportation Systems (ITS); Security; Security header and certificate formats.
- [B8] FIPS Pub 140-2, Security requirements for Cryptographic Modules, Federal Information Processing Standards Publication 140-2, U.S. Department of Commerce/N.I.S.T., Springfield, VA, June 2001 (supersedes FIPS Pub 140-1).
- [B9] Goldberg, I. and D. Wagner, “Randomness and the Netscape Browser: How Secure Is the World Wide Web?” *Doctor Dobb’s Journal*, Jan. 1996. Available: <http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html>.
- [B10] Howgrave-Graham, N. A. and N. P. Smart, “Lattice Attacks on Digital Signature Schemes,” *Designs, Codes and Cryptography*, vol. 23, no. 3, pp. 283–290, Aug. 2001.
- [B11] IEEE Std 802.11™, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.<sup>14, 15</sup>
- [B12] IEEE Std 1609.4™-2010, Draft Standard for Wireless Access in Vehicular Environments (WAVE)—Multi-Channel Operation.
- [B13] IETF Request for Comments: 5246, The Transport Layer Security (TLS) Protocol Version 1.2.<sup>16</sup>
- [B14] ISO 3166-2:2013, Codes for the representation of names of countries and their subdivisions—Part 2: Country subdivision code.<sup>17</sup>

---

<sup>14</sup> The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

<sup>15</sup> IEEE publications are available from The Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (<http://standards.ieee.org/>).

<sup>16</sup> IETF documents (i.e., RFCs) are available for download at <http://www.rfc-archive.org/>.

- [B15] National Institute for Standards and Technology (NIST) Special Publication (SP) 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Jan. 2012. Available: <http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf>.
- [B16] Nguyen, P. and I. Shparlinski, “The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces,” *Designs, Codes and Cryptography*, Vol. 30, No. 2, pp. 201–217, 2003.
- [B17] OASIS (Organization for the Advancement of Structured Information Standards), PKCS #11 Cryptographic Token Interface Standard. Available: [https://www.oasis-open.org/committees/documents.php?wg\\_abbrev=pkcs11](https://www.oasis-open.org/committees/documents.php?wg_abbrev=pkcs11).
- [B18] Pintsov, L. A. and S. A. Vanstone, “Postal revenue collection in the digital age,” *Financial Cryptology*, pp. 105–120, Oct. 26, 2001.
- [B19] Preciosa Project, PRECIOSA—Privacy Enabled Capability in Co-operative Systems and Safety Applications.
- [B20] SAE J2735, Dedicated Short Range Communications (DSRC) Message Set Dictionary.<sup>18</sup>
- [B21] SAE J2945/1, On-Board System Requirements for V2V Safety Communications.
- [B22] SeVeCom—Secure Vehicle Communication, Deliverable 1.1, VANETS Security Requirements Final Version.
- [B23] Software in the Public Interest, Inc. Debian security advisory, DSA-1571-1openssl—predictable random number generator, 2008. Available: <http://www.debian.org/security/2008/dsa-1571>.
- [B24] Whyte, W., A. Weimerskirch, V. Kumar, and T. Hehn, “A security credential management system for V2V communications,” *2013 IEEE Vehicular Networking Conference (VNC)*, pp. 1–8, Dec. 6–18, 2013.

---

<sup>17</sup> ISO publications are available from the ISO Central Secretariat (<http://www.iso.org/>). ISO publications are also available in the United States from the American National Standards Institute (<http://www.ansi.org/>).

<sup>18</sup> SAE publications are available from the Society of Automotive Engineers (<http://www.sae.org/>).