IEEE Std 1609.2™-20XX
(Amendment of
IEEE Std 1609.2-2016)

“IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages” – consolidated draft of IEEE 1609.2-2016 with amendments specified in 1609.2a / D8

Sponsor

Intelligent Transportation Systems Committee
of the
IEEE Vehicular Technology Society

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

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

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, Kevin Smith, William Whyte, *Vice Chairs*

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

When the IEEE-SA Standards Board approved this standard onXXX, it had the following membership:

*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.

This amendment addresses multiple needs to enhance and extend IEEE Std 1609.2-2016:

- Since the publication of Standard 1609.2-2016, a number of errors, omissions and ambiguities have been discovered, which this amendment corrects.

- Industry stakeholders have requested additional functionality, in particular better support for compact expressions of ranges of Service Specific Permissions (SSPs).

- Test vectors are provided to enable implementers to gain confidence in correctness of their implementation before running interoperability tests.

- Additional informative material is provided to assist implementers of the standard and users of the security services in understanding the intended implementation and use.

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 3

3. Definitions, abbreviations, and acronyms 4
3.1 Definitions 4
3.2 Abbreviations and acronyms 9

4. General description 11
4.1 WAVE protocol stack overview 11
4.2 Secure data service (SDS) 14
4.3 Security services management entity (SSME) 18
4.4 Behavior of SDEEs 19

5. Cryptographic operations and validity 20
5.1 Certificate validity 20
5.2 Signed SPDU validity 33
5.3 Cryptographic operations 45

6. Data structures 50
6.1 Presentation and encoding 50
6.2 Basic types 50
6.3 Secured protocol data units (SPDUs) 51
6.4 Certificates and other security management data structures 66

7. Certificate revocation lists (CRLs) and the CRL Verification Entity 82
7.1 General 82
7.2 CRL Verification Entity specification 82
7.3 Data structures 83
7.4 CRL: 1609.2 Security envelope 88

8. Peer-to-peer certificate distribution (P2PCD) 92
8.1 General 92
8.2 P2PCD operations 93
8.3 P2PCD Entity specification 105
8.4 Data structures 106

9. Service primitives and functions 108
9.1 General comments and conventions 108
9.2 Identifiers used in the interface specification 110
9.3 Sec SAP 115
9.4 SSME SAP 146
9.5 SSME-Sec SAP 166

Annex A (normative) Protocol Implementation Conformance Statement (PICS) proforma 171
A.1 Instructions for completing the PICS proforma 171
A.2 PICS proforma—IEEE Std 1609.2 173

Annex B (normative) ASN.1 modules....184
B.1 General....184
B.2 1609.2 security services....184
B.3 Certificate revocation list (CRL)....194
B.4 Peer-to-peer certificate distribution (P2PCD)....198

Annex C (informative) Specifying the use of IEEE Std 1609.2 by SDEEs....200
C.1 General....200
C.2 IEEE 1609.2 security profiles....200
C.3 IEEE 1609.2 security profile proforma....211
C.4 Service Specific Permissions (SSP)....213
C.5 Assurance level....214
C.6 Recommendations on certificates....214
C.7 Source of encryption keys....215

Annex D (informative) Examples and use cases....217
D.1 Guidance for SDEE specifiers and implementers....217
D.2 Processing CRLs....218
D.3 Constructing a certificate chain....219
D.4 Peer-to-peer certificate distribution....224
D.5 Example data structures....231
D.6 Cryptographic Test Vectors....236

Annex E (informative) Deployment considerations....262

Annex F (informative) Bibliography....264

“IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages” – consolidated draft of IEEE 1609.2-2016 with amendments specified in 1609.2a

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][1], 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.

[1] The numbers in brackets correspond to those of the bibliography in Annex F.

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.

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.[2]

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.[3, 4]

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.[5]

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

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

Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography," Version 2.0, May 21, 2009.[7]

[2] 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/).
[3] IEEE publications are available from The Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).
[4] The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.
[5] IETF publications are available from http://www.ietf.org.
[6] NIST special publications are available from http://csrc.nist.gov/publications/nistpubs/.
[7] SECG publications are available from http://www.secg.org.

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.

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.[8]

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.

[8] *IEEE Standards Dictionary Online* subscription is available at: http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.

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**.

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 certficate'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**.

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: An entity that is requesting certificates or signing Protocol Data Units. *Contrast*: **certificate authority**. Note that an entity may act as an end-entity in one context and as a Certificate Authority in another context.

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

enrollment 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.**

length of certificate chain: (In a certificate chain from any CA certificate to a final certificate, with each certificate in the chain issuing the one after it): the number of certificates in the chain, except for the first one; in other words, the number of certificates in the chain, minus 1.

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 SDEEs receiving information that could cause them to take incorrect actions.

misbehavior authority: A management component on the network with responsibility for determining which SDEEs 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.

peer-to-peer certificate distribution: A mechanism for allowing WAVE Security Services instances to learn certificates from peer instances.

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 WAVE 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*.

whole-certificate hash algorithm: the algorithm used to calculate the hash of a certificate for purposes of identifying that certificate.

Wireless Access in Vehicular Environments (WAVE) device: A device that is compliant to IEEE Std 1609.3, IEEE Std 1609.4™, 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 MessagePprotocol (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.

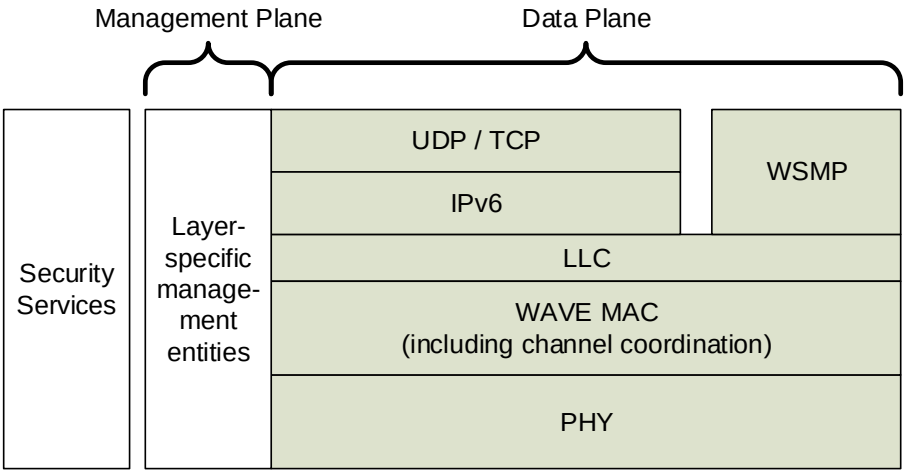


Figure 1—WAVE reference model

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 s*ecure 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.

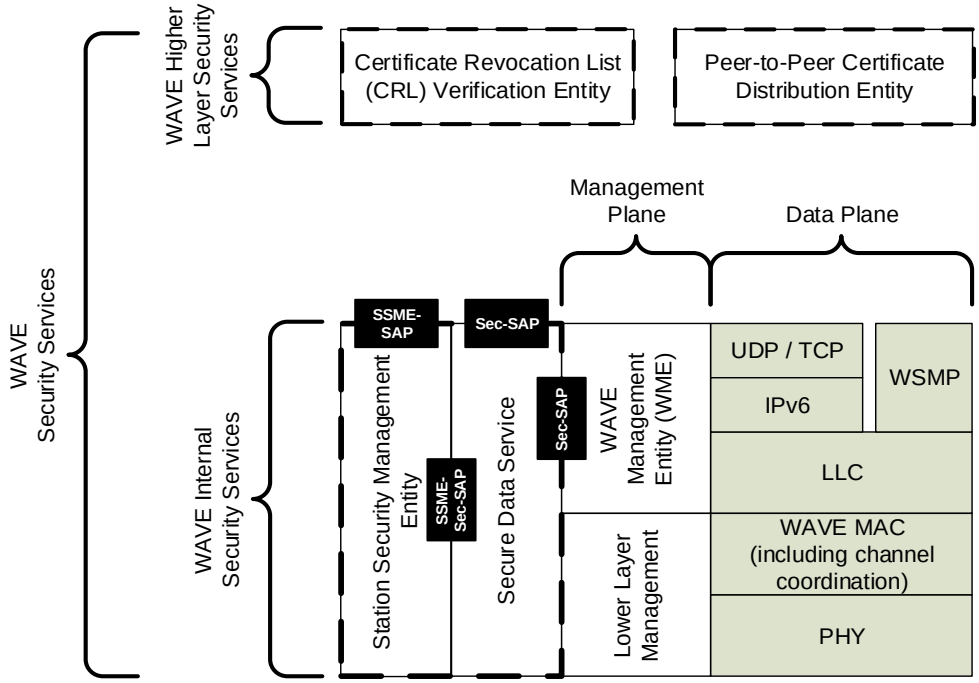


Figure 2—WAVE protocol stack showing detail of WAVE Security Services

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.

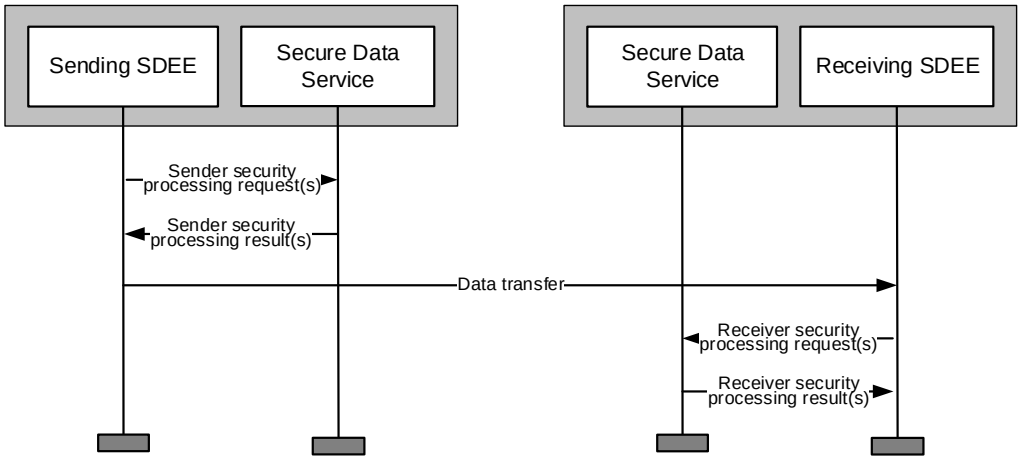


Figure 3 —Process flow for use of 1609.2 secure data service

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

- Generate signed SPDU (see 4.2.2.2.3)
- Verify signed SPDU (see 4.2.2.3.2)
- Generate encrypted SPDU (see 4.2.2.2.4)
- Decrypt encrypted SPDU (see 4.2.2.3.3)
- The CRL Verification Entity (see 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 6.3.25.

Otherwise, provision of this service is optional.

The SDS is requested to preprocess a received SPDU 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 SPDU

— If the SPDU was of type signed:

  — The Service Specific Permissions of the signer as defined in 5.2.3.3.3

  — The geographic validity region of the signer’s certificate as defined in 6.4.17

  — 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 SPDU.

The SDS is requested to verify a signed SPDU 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 SPDU 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 SPDU.

The SDS is requested to decrypt an encrypted SPDU 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 SPDU that is the correct decryption of the ciphertext within the encrypted SPDU.

— 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* (occasionally referred to in this document simply as the “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. Additionally, the security profile allows the SDEE specifier to specify other aspects of the security behavior of the SDEE; see Annex C for more information.

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-Verify-Certificate.confirm

— To request deletion of information about a certificate via SSME-DeleteCertificate.request 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-AddRevocationInfo.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.

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 6.4.31 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. 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 *enrollment 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*.[9] 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.

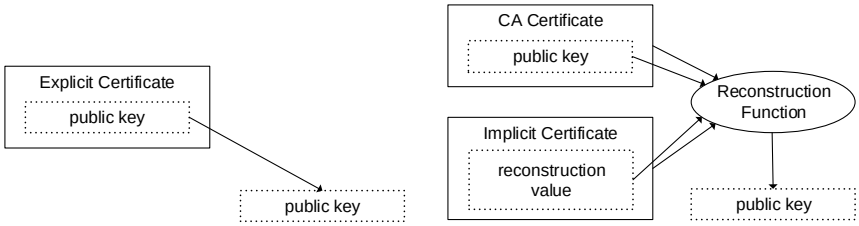


Figure 4—Implicit and explicit certificates

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", (equivalently, "first" to "last" or "beginning" to "end") such that each certificate in the chain, except the last one, is the *issuing certificate* for one below (or "after") it and each certificate, except the first one, is the *subordinate certificate* of the certificate above (or "before") 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 standard uses the terminology "B

[9] Elliptic Curve Qu-Vanstone (ECQV) or "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].

issues A" even though it would be more correct to use the terminology "B's holder issues A" or "the private key associated with B 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. A certificate chain that starts with a root certificate is referred to in this standard as a *full certificate chain*.

The *length of a certificate chain* is defined as the number of certificates in the chain apart from the topmost one, or equivalently as the number of intra-certificate gaps in the chain. Figure 5 illustrates these two definitions. Figure 6 illustrates that the definition also applies to a "subchain" within a longer chain, i.e. that the definition does not assume that the topmost certificate is a root certificate or that the bottom certificate is an end-entity certificate.

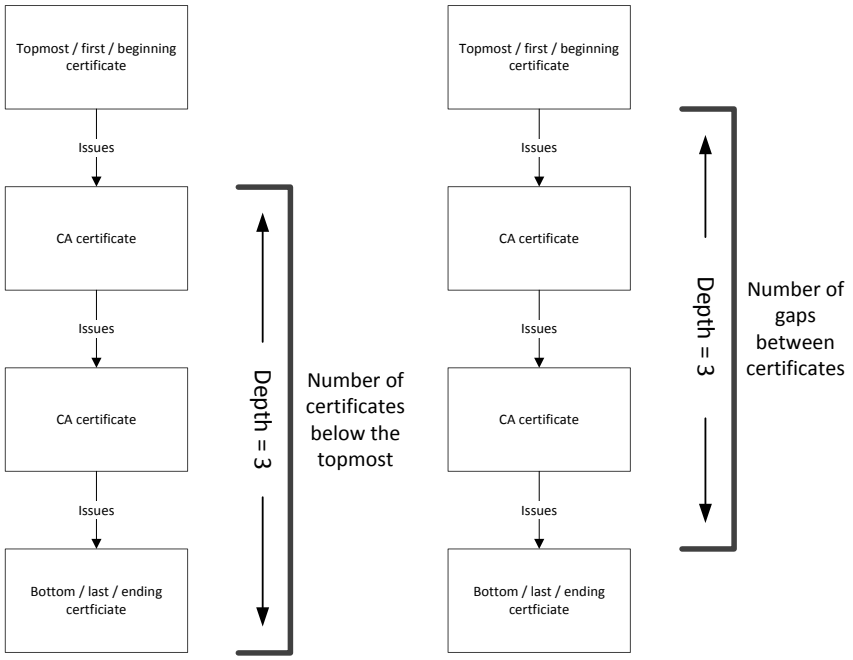


Figure 5—A certificate chain of length 3

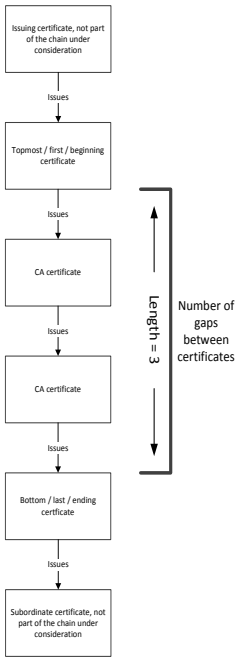


Figure 6—A subchain of length 3 within a longer chain

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 SPDU 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 SPDU comes to be validated, two chains can be constructed consistent with the issuer identifiers. The certificate or SPDU 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 full certificate chain that it can support. A conformant implementation shall support a maximum length of at least two, i.e. a maximum total number of certificates in the chain of at least three. 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 7 for an illustration.

— If the chain ends with one or more implicit certificate: the signature on a signed SPDU 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 SPDU 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 7 for an illustration involving explicit certificates. See Figure 8 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.

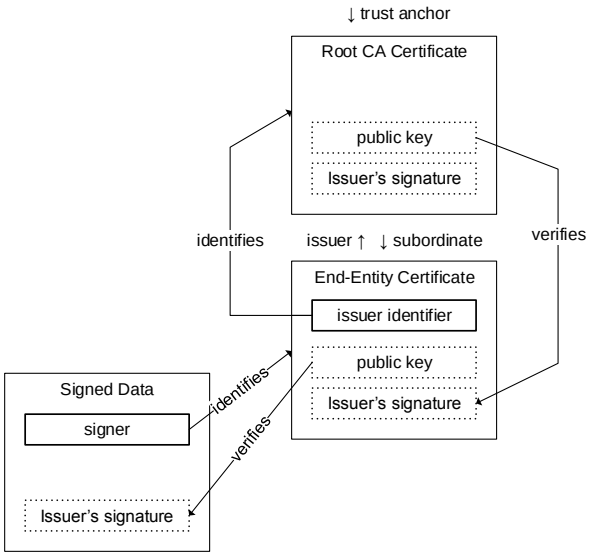


Figure 7— Cryptographic verification of a signed SPDU with a full certificate chain, using explicit certificates

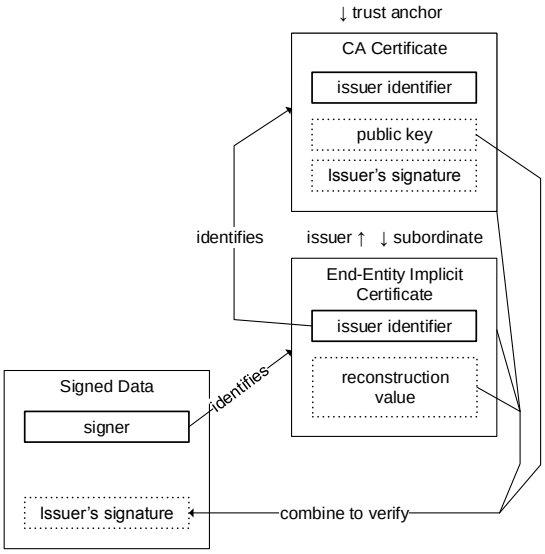


Figure 8—Cryptographic Verification of a signed SPDU with a (non-full) certificate chain with implicit end-entity certificate

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 self-signed certificate (i.e., a root CA certificate) is consistent with itself by definition. 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 issung entry. See 6.4.35 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, the permitted length of the chain in the subordinate entry is consistent with the permitted length of the chain in the issuing entry. Specifically, if *minChainLength* and *chainLengthRange* in the subordinate certificate and issuing certificate

have the values mcl_s, clr_s, mcl_i, clr_i, respectively, then $mcl_i \leq mcl_s+1$ and $(mcl_i + clr_i) \geq (mcl_s + clr_s + 1)$. (In the case where the subordinate certificate is an end-entity certificate, mcl_s and clr_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:

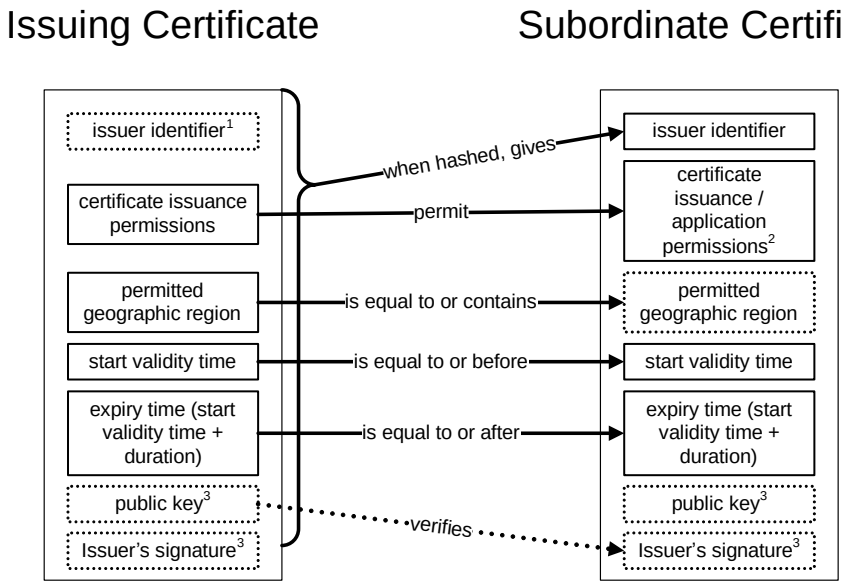
    — All the values of `eeType` in the subordinate entry also appear in the issuing entry.

— **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 9 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.



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 9—Consistency of permissions between an issuing and a subordinate certificate, and within the subordinate certificate

5.1.2.5 Trustworthiness of a certificate chain

A certificate chain is trustworthy if the following hold:

— Each subordinate certificate is consistent with its issuing certificate.

— The chain begins with a trust anchor.

— None of the certificates in the chain have been revoked, as discussed in 5.1.3.

— (Optional) none of the certificates in the chain have expired at the time of chain verification, as discussed in 5.2.4.2.1 and 5.2.4.2.7.

Whether the expiry test is applied is specified as part of the SDEE specification, as discussed in 5.2.4.2.7. It is strongly recommended that chains with expired certificates are treated as untrustworthy.

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 certificate and first received after the issue date of the revocation statement are revoked, even if their stated start of validity period 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 authority with authorization to issue revocation information for that certificate. Each such authority might be entitled to issue and keep up to date more than one set of revocation information, but there is one specific set that is identified as the one that will contain revocation information about C if it is revoked. Therefore, 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 c*ertificate 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 that certificate's *Certificate Revocation Authorizing CA* (CRACA) certificate, and
— Indicates that certificate's *CRL Series* value, and
— Is of the appropriate type (linkage-based or hash ID-based)

A 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 one of the certificates in *C*'s full 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.

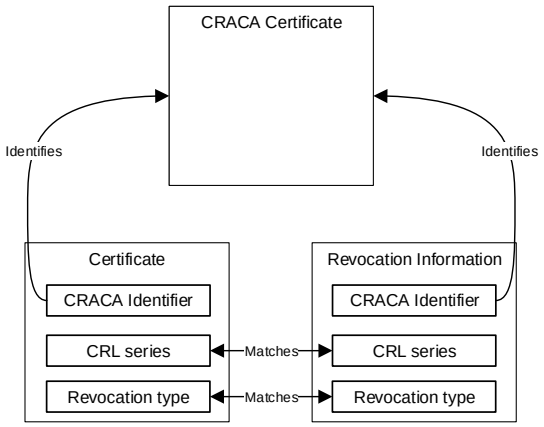


Figure 10—Relationship between certificate and revocation information

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 11.

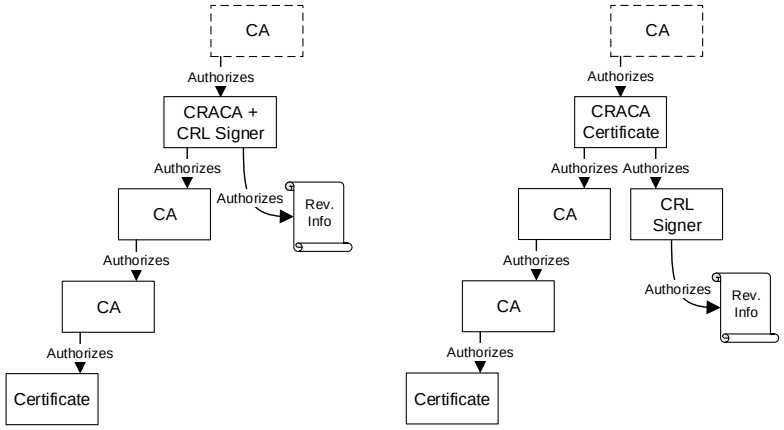


Figure 11—Revocation information: issued by a CRACA (on the left) or by a CRL signer directly authorized by the CRACA (on the right)

5.1.3.3 Identification of CRACA certificate

A certificate contains a *cracaId* field as specified in 6.4.8. This is an octet string of length 3. The relevant CRACA certificate is the certificate in the full chain for which the low-order three bytes of its SHA-256 hash are equal to the *cracaId*. The hash of the certificate is obtained as specified in 6.3.26.

A *cracaId* 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 the certificate has a very short lifetime or for some other reason.

A *cracaId* 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 *cracaId*, and the *cracaId* is not matched by a unique certificate in the full chain (i.e either it is not matched at all, or it is matched by more than one certificate), then the certificate is invalid.

Figure 12 illustrates logic flows used in determining the CRACA for a certificate.

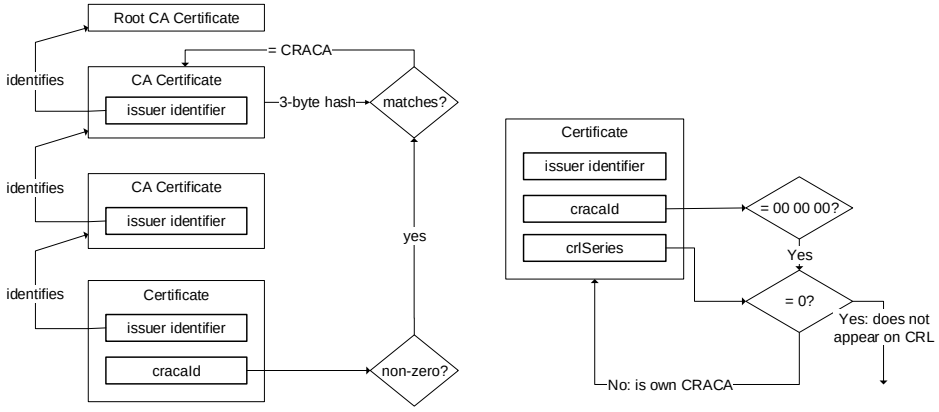


Figure 12—Examples: determining the CRACA certificate for a certificate

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 an SDEE 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 SDEE to be revoked. *Group linkage information* allows certificates owned by all SDEEs 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* || *LinkageSeed1* || 0^{112})], where 0^{112} is 112 0 bits

   2) Set *LinkageSeed2* = the low-order 16 octets of [SHA-256(*LinkageAuthorityIdentifier2* || *LinkageSeed2* || 0^{112})]

   3) Set *iRev* = *iRev*+1

c) For *j* = 0 to *jMax* – 1:

   1) Set *data* = *LinkageAuthorityIdentifier1* || Uint32(*j*) || 0^{80}, where Uint32(*j*) indicates *j* represented as a 4-octet integer in network byte order

   2) Set *PreLinkageValue1*(*j*) = AES (key = *LinkageSeed1*, data = *data*) XOR (*data*), where *LinkageSeed1* is the value *LinkageSeed1* takes after completing the iterative process defined in step b)

   3) Set *PreLinkageValue2*(*j*) = AES (key = *LinkageSeed2*, data = [0^{10} || *LinkageAuthorityIdentifier2* || Uint32(*j*)]) XOR [0^{10} || *LinkageAuthorityIdentifier2* || 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 certifiate 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* || *GroupLinkageSeed1* || 0^{112})], where 0^{112} is 112 0 bits

   2) Set *GroupLinkageSeed2* = the low-order 16 octets of [SHA-256(*LinkageAuthorityIdentifier2* || *GroupLinkageSeed2* || 0^{112})]

   3) Set *iRev* = *iRev* + 1

c) Set *data* = *LinkageAuthorityIdentifier1* || Uint32(*j*) || 0^{80}, where Uint32(*j*) indicates *j* represented as a 4-octet integer in network byte order.

d) Set *PreLinkageValue1* = AES (key = *GroupLinkageSeed1*, data = *data*) XOR *data*, where *GroupLinkageSeed1* is the value *GroupLinkageSeed1* takes after completing the iterative process defined in step b)

e) Set *PreLinkageValue2* = AES (key = *GroupLinkageSeed2*, data = [*LinkageAuthorityIdentifier2* || Uint32(*j*) || 0^{80}]) XOR [*LinkageAuthorityIdentifier2* || Uint32(*j*) || 0^{80}]

f) Set *GroupLinkageValue* = the low-order 9 bytes of *PreLinkageValue1* 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 full chain associated with a signed SPDU might potentially be dubious. The primitive Sec-SecureDataPreprocessing.confirm indicates the earliest `nextCrl` time associated with any certificate in the full 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 full 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 full 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

The SDS supports two forms of signed SPDU: SPDUs signed by a certificate, and self-signed SPDUs. The 1609.2 security profile (see Annex C) is provided to enable an SDEE specification to state which form or forms are permitted for that SDEE. It is strongly recommended that SDEE specifications permit only SPDUs signed by a certificate.

A signed SPDU 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.

Figure 13 illustrates the different validity conditions used to determine the validity of a signed SPDU that is signed by a certificate, and the input information used to check against those validity conditions. Figure 14 shows the information fields that go into creating a signed SPDU that is signed by a certificate.

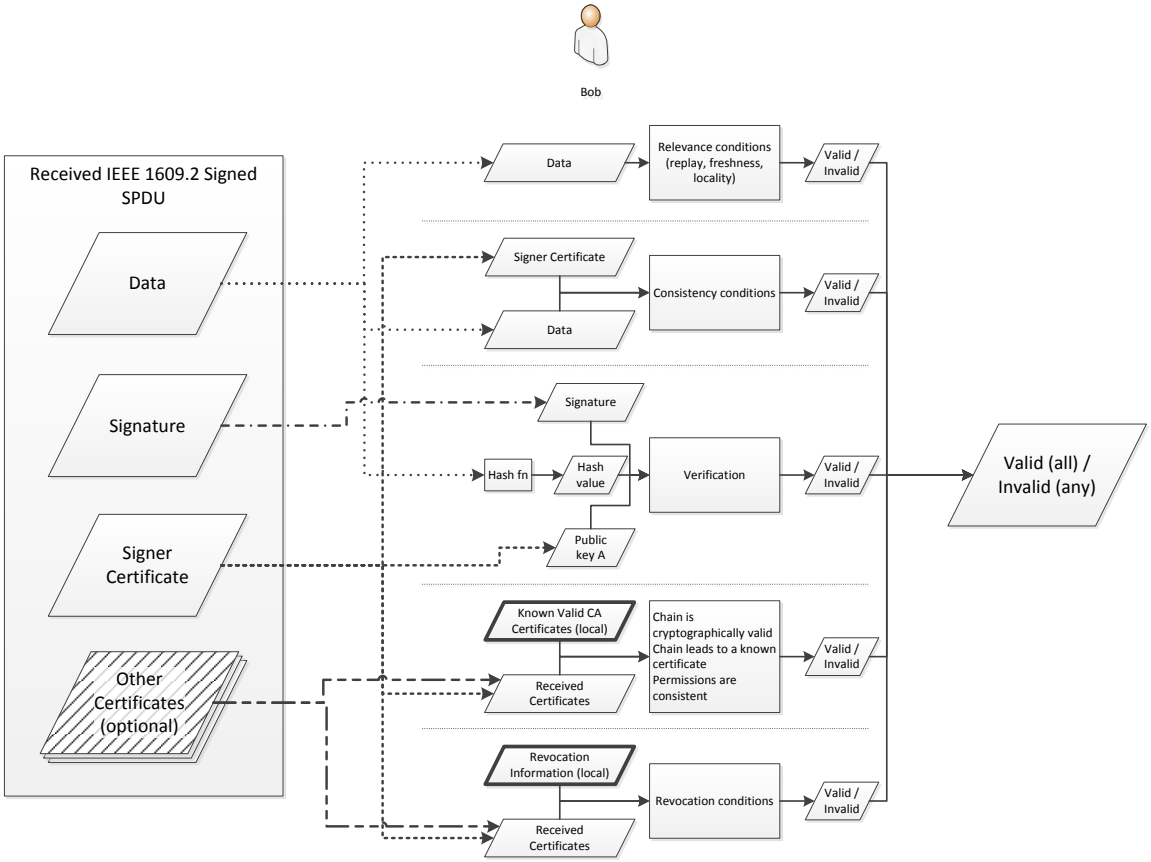


Figure 13—Validity conditions for a signed SPDU

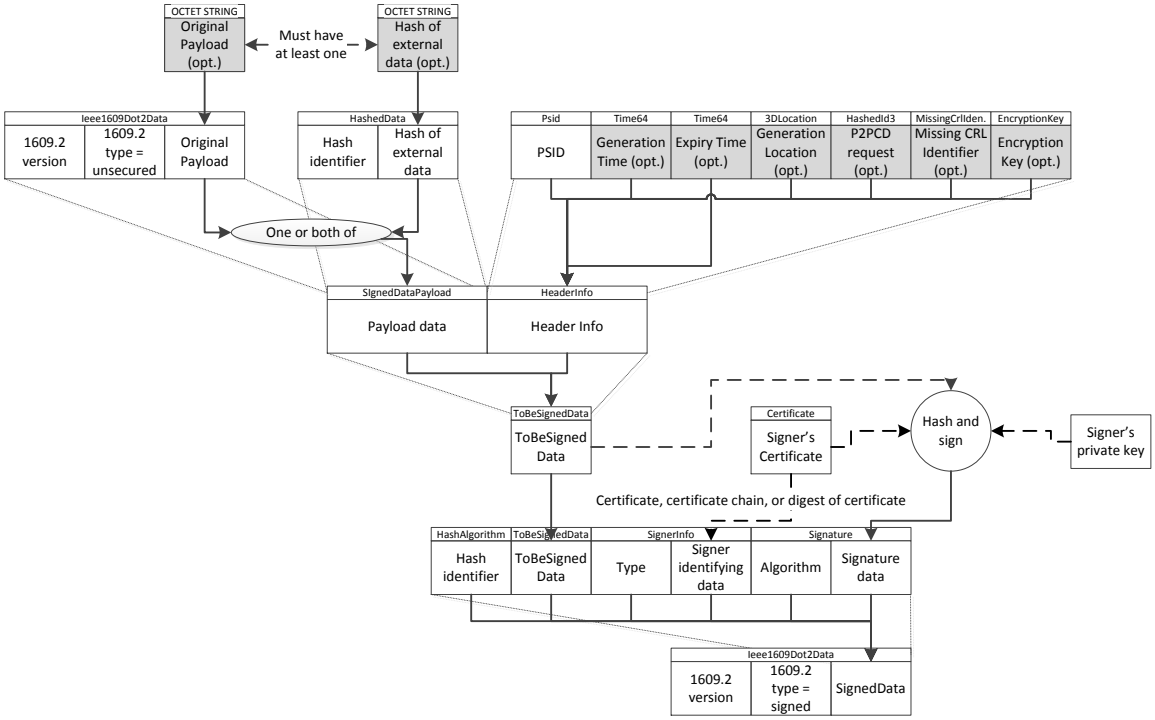


Figure 14—Decomposition of a signed SPDU

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 SPDU is correctly formed using the data structures of Clause 6.
- The signature on the SPDU verifies with the appropriate certificate or public key, as specified in 5.2.3.2.2.
- The SPDU is internally consistent, as specified in 5.2.3.2.4.
- EITHER the SPDU is self-signed and the SDEE specification permits self-signed SPDUs;
- OR the SPDU is signed with a certificate and all of the following conditions hold:
    - 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 permssions in the signing certificate, and the security envelope is consistent with itself, as specified in 5.2.3.2.3.

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.If a signed SPDU does not meet the first three conditions and either the fourth or the fifth, it is invalid.

5.2.3.2.2 Signature verification

If the signature on a signed SPDU does not pass cryptographic verification, the SPDU is invalid. Signature generation and verification is specified in 5.3.1.

In the case of a signed SPDU signed with a certificate, the certificate to use to verify the signature is indicated using the SignerIdentifier structure within the SignedData as specified in Clause 6.

In the case of a self-signed SPDU, the public key is not transported in the IEEE 1609.2 security envelope. In this case, the means by which the receiving SDEE obtains the public key are part of the SDEE specification.

5.2.3.2.3 Consistency between signed SPDU and signing certificate

A signed SPDU that is signed with a certifiate 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, if present, is consistent with~~in~~ the geographic validity 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 generation time is available (either from the headers of the signed SPDU, or obtained by other means such as from the SPDU payload by the SDEE and provided to the SDS) and is within the validity period of the certificate.

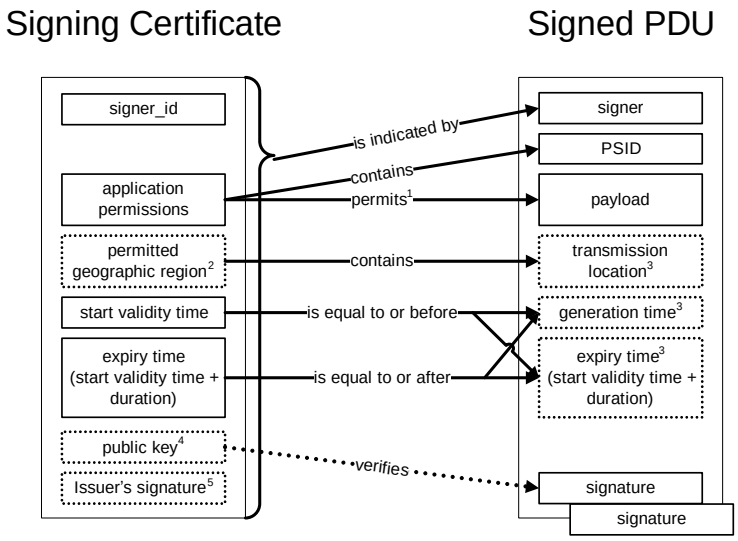
— The expiry time, if present, is within the validity period of the certificate.

— 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 15 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.



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.

Figure 15—Consistency of permissions between signed SPDU and signing certificate

5.2.3.2.4 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.
- (If signed with a certificate) 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.
- (If signed with a certificate) The number of certificates in the full chain ending in the SPDU-signing certificate is less than some SDEE-specific limit (see Annex C). This condition can be verified by the SDS.
- (If signed with a ertificate) If the signed SPDU is making a statement about a geographic region other than a single point, that region is contained within the validity region of the certificate as specified in 5.2.3.3.5. This condition is not verified by the SDS and is intended to be verified by the receiving SDEE.

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 15, 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 that is signed by a certificate 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.

5.2.3.3.5 Consistency between SPDU payload and permissions: Relevance region

If a signed SPDU was signed with a certificate, the generation location consistency conditions specified in 5.2.3.2.3 can be used to determine that a signed SPDU was generated in a location where the generating SDEE is permitted to operate. However, depending on the application use case, generation location might not need to be the subject of a consistency check, and there also might be other geographic information for which it is appropriate to require authorization. An example of generation location not needing to be checked is Certificate Revocation List (CRL) generation activity following the specification in clause 7; in this case, the location at which the CRL is generated is not germane to whether or not the CRL is valid, and the CRL could in fact be generated in a physical location that is outside any region that the revoked certificates would have been valid in. An example of other geographic information for which it is appropriate to require authorization is given by the Signal Phase and Timing (SPaT) message defined in SAE J2735 [B20]. In this case, the SPaT message can include information about signal phase and timing at multiple intersections, and it is appropriate to require that all the locations about which the message makes statements are permitted by the certificate.

Consistency between relevance areas in the payload and the certificate is SDEE-specific and outside the scope of this standard. The Sec-SecureDataPreprocessing.request returns the geographic region associated with a certificate, and the SDEE is expected to carry out any consistency checks necessary to determine that the payload is consistent with that geographic region. The 1609.2 security profile specified in Annex C can be used to note that additional geographic consistency checks are to be carried out; however, the details of these geographic consistency checks should be defined as part of the specification of the SDEE.

5.2.3.4 Identified Regions

If a signed SPDU was signed with a certificate, then per the consistency conditions specified in 5.1.2.4, 5.2.3.2.3, and 5.2.3.3.5, in a valid signed SPDU both of the conditions below hold:

— The geographic validity regions in each subordinate certificate are consistent within the chain, meaning that each validity region in a subordinate certificate is wholly contained in the validity region of its issuing certificate.

— If so specified by the SDEE specification, the generation location or relevance region of the SPDU is consistent with the validity region of the SDEE's certificate, meaning that the generation location or relevance region is respectively inside or wholly contained within that validity region.

This standard allows multiple approaches to indicate a validity region in a certificate. One of these approaches is to include an identifier for the region in the certificate, such that the SDS maps from the region identifier to a representation of the region which may be used for validity checking. The accuracy of this representation is addressed below in this subclause.

The IdentifiedRegion identifier may be drawn from one of a number of dictionaries. The permitted dictionaries are specified in 6.4.22 and the subclauses immediately thereafter.

The Protocol Implementation Conformance Statement (PICS) proforma given in Annex A allows the vendor of an implementation of WAVE Security Services to state whether any identified regions are supported, and to indicate which particular regions are supported in the sense that the WAVE Security Services have access to a map from that identifier to a region representation.

In claiming support of a particular region identifier *RId*, contained in one of the supported dictionaries and representing a region *R*, the PICS for an implementation is indicating that the following conditions hold:

— The region representation for *R* enables all consistency conditions with respect to identifiers in the same dictionary to be carried out with respect to *R*, i.e.:

  — In addition to supporting *RId* in this sense, the implementation supports all identifiers in the dictionary that identify a region that fully contains *R*.

  — For each region that fully contains *R*, the representation of the containing region fully contains the representation of *R*.

  — For each region that does not fully contain *R*, the representation of that region does not fully contain the representation of *R*.

The 1609.2 security profile is provided to enable SDEE specifiers to specify whether the SDEE should use the identified region type, and if so the representation accuracy requirements that apply (see Annex C).

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 an 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 received by a given SDEE is valid only if it is valid with respect to each of the relevance conditions appropriate to that SDEE. The possible relevance conditions are as follows.

— **Freshness**: The signature generation time is not too far in the past, relative to the receiving SDS's estimate of the current time, for some definition of "too far in the past" given in the SDEE specification.

— **Future generation**: The signature generation time is not (too far) in the future, relative to the receiving SDS's estimate of the current time, for some definition of "too far in the future" given in the SDEE specification.

— **Expiry**: The signed data has not expired relative to the receiving SDS's estimate of the current time. Whether or not to carry out this check is specified in the SDEE specification.

— **Location**: The generation location is not too far away from the receiving SDS's estimate of its location, for some definition of "too far away" given in the SDEE specification.

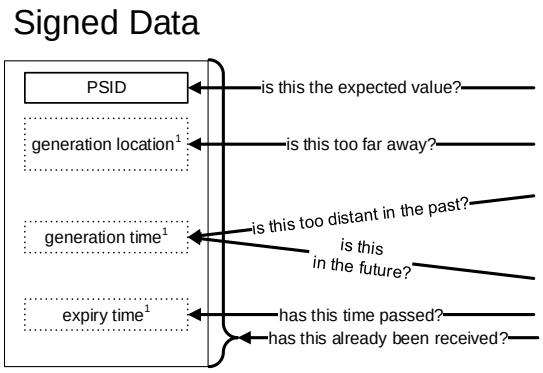
— **Replay**: The PDU is not a replay of a PDU acted upon by that SDEE in the recent past, for some PSID-specific definition of "the recent past". Whether or not to carry out this check is specified in the SDEE specification.

— **Certificate expiry**: For an SPDU signed with a certificate, none of the certificates in the full chain ending with the certificate that signed the signed SPDU have expired relative to the receiving SDS's estimate of the current time. Whether or not to carry out this check is specified in the SDEE specification.

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 16 illustrates the fields to which relevance tests apply (the check for expired certificates is not illustrated in Figure 16). The relevance and replay tests to be carried out are specified as inputs to Sec-SignedDataVerification.request.



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

Figure 16—Replay and relevance tests

5.2.4.2.2 Generation time too far in the past

The SDS provides the service of checking whether a signed SPDU received by an SDEE has a generation time too far in the past. The 1609.2 security profile (see Annex C) is provided to enable an SDEE specification to state whether this service is used by that SDEE and, if so, to provide a definition of "too far in the past".

The SDS uses the following algorithm to check for a signed SPDU having been generated too far in the past.

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 SDS provides the service of checking whether a signed SPDU received by an SDEE has a generation time too far in the future. The 1609.2 security profile (see Annex C) is provided to enable an SDEE specification to state whether this service is used by that SDEE and, if so, to provide a definition of "too far in the future".

The SDS uses the following algorithm to check for a signed SPDU having been generated too far in the future.

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 SDS provides the service of checking whether a signed SPDU received by an SDEE has passed some expiry time stated in the SPDU. The 1609.2 security profile (see Annex C) is provided to enable an SDEE specification to state whether this service is used by that SDEE.

The SDS uses the following algorithm to check for a signed SPDU having 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 SDS provides the service of checking whether a signed SPDU received by an SDEE was generated too far away. The 1609.2 security profile (see Annex C) is provided to enable an SDEE specification to state whether this service is used by that SDEE and, if so, to provide the definition of "too far away".

The SDS uses the following algorithm to check for a generation location being 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 provides the service of checking whether a signed SPDU received by an SDEE is a replay, i.e. whether it is a duplicate of a signed SPDU recently processed by the SDS for that SDEE. The 1609.2 security profile (see Annex C) is provided to enable an SDEE specification to state whether this service is used by that SDEE. The definition of "recently processed" 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 5.2.4.2.1), and the interfaces defined in this standard enforce that the two values are the same.

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 BOTH the COER encoding of the tbsData field canonicalized according to the encoding considerations given in IEEE 1609.2 clause 6.3.6, AND the COER encoding of the Certificate that is to be used to verify the SPDU, canonicalized according to the encoding considerations given in clause 6.4.3, are identical to those information elements for another recently received SPDU.

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.

5.2.4.2.7 Certificate expiry

For an SPDU signed with a certificate, the SDS provides the service of checking whether any certificate in the chain of that signed SPDU has expired at the time the SPDU is verified. The 1609.2 security profile (see Annex C) is provided to enable an SDEE specification to state whether this service is used by that SDEE. Annex C also provides discussion of how to establish whether certificate expiry detection is important for a particular SDEE.

The SDS uses the following algorithm to determine whether the certificates in the chain of 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 full 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.

NOTE-- Certificates that have expired are risky to trust. It is strongly recommended that a signed SPDU received after the expiry time of any certificate in its full chain be rejected.

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 interpret 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 interpret 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), 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.[10]

Three elliptic curves are specified for use with ECDSA: NIST P-256 as specified in FIPS 186-4, and (brainpoolP256r1, brainpoolP384r1) 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. A conformant implementation that supports signing or verification shall support at least one of these curves and may support more.

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”).

[10] The additional information may allow faster verification operations as described in SEC1 Version 2 and Antipa, et al. [B2].

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
    i) If the verification type is *certificate*, *signer identifier input* shall be the certificate with which the message is to be verified, canonicalized as specified in 6.4.3.
    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 6.3.25.

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.[11]

5.3.2 Implicit certificates

In this standard, implicit certificates are processed as specified in Standards for Efficient Cryptography (SEC) 4 with the exceptions noted in this subclause.

a) In this standard, an implicit certificate is 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 in the instance the implicit certificate is hashed to an integer modulo 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, canonicalized as specified in 6.4.3) || Hash (Entirety of issuer certificate, canonicalized as specified in 6.4.3).

c) SHA-256 shall be used as the Hash algorithm H.

[11] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

d) Within the integer hash function Hn, the output of the hash function H is not converted to an integer mod n using the mechanism specified in SEC 4, section 2.3. Instead, the hash function is converted to an integer by taking the 256-bit output from SHA-256, converting that bit string to an octet string using the Bit String To Octet String Conversion Primitive of SEC 1, and then converting that octet string to an integer using the Octet String To Integer Conversion Primitive of SEC 1.

This standard defines implicit certificates over the curves NIST P-256 and brainpoolP256r1. This standard does not define certificates over the curve brainpoolP384r1.

SHA-256 shall be used as the Hash algorithm H used by 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.

5.3.3 Hash algorithms: SHA-256, SHA-384

The hash algorithms approved for use in this standard are SHA-256 and SHA-384 as specified in the Federal Information Processing Standard (FIPS) 180-4. In this standard, the phrase "the SHA-256 (resp. SHA-384) hash of [an octet string]" is used to mean "the hash of [that octet string] obtained using SHA-256 (resp. SHA-384) 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 6.3.33.
- 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. This standard supports the use of ECIES to encrypt ephemeral data encryption keys as specified in 5.3.4.1 and does not support the use of ECIES to encrypt data directly.

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.

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

NOTE—IEEE Std 1363a specifies the use of ECIES to encrypt data; in this standard, as noted above, ECIES is used only to encrypt symmetric keys. In the bulleted list below the word “data” is used to describe the plaintext input to encryption for consistency with IEEE Std 1363a, even though in the case of this standard the input is in fact a key.

a) The secret value derivation primitive shall be Elliptic Curve Secret Value Derivation Primitive–Diffie-Hellman version with cofactor multiplication (ECSVDP-DHC).

b) Compatibility with the corresponding –DH primitive shall not be desired.

c) 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

d) 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

e) Encryption shall use non-DHAES mode.

f) 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 used.

The ephemeral public key *V* shall be freshly generated for each encryption operation, i.e. an encryption operation shall not reuse an ephemeral public key *V*.

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*), where *c* is the COER encoding of the certificate, canonicalized per 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*), where *d* is the COER encoding of the Ieee1609Dot2Data, canonicalized per 6.3.4.

— 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. See Annex C.7 for guidance on when different approaches to obtaining the encryption key may be appropriate. The data structures in clause 6 allow the sender of an encrypted message to indicate the source of the encryption key to the recipient.

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.

Example test vectors for ECIES are provided in D.6.2.

Example test vectors for MAC1 are provided in D.6.3.

Example test vectors for KDF2 are provided in D.6.4.

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™-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.

Example AES-CCM test vectors are provided in D.6.1

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.. Any structure for which encoding is subject to canonicalization has a paragraph entitled **Encoding considerations** in its description in Clause 6.

The complete IEEE 1609.2 ASN.1 modules are given in Annex B. In the event of a conflict between Annex B and this clause, this clause takes precedence.

6.2 Basic 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))
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 6.3.32. The subclauses between 6.3.4 and 6.3.32 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.

Critical information fields: This is not a critical information field as defined in 5.2.5.

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.

  — If `signer` indicates the choice `self`, then the signature calculation is parameterized as follows:

    — *Data input* is equal to the COER encoding of the `tbsData` field canonicalized according to the encoding considerations given in 6.3.6.

    — *Verification type* is equal to *self*.

    — *Signer identifier input* is equal to the empty string.

  — If `signer` indicates `certificate` or `digest`, then the signature calculation is parameterized as follows:

    — *Data input* is equal to the COER encoding of the `tbsData` field canonicalized according to the encoding considerations given in 6.3.6.

    — *Verification type* is equal to *certificate*.

    — *Signer identifier input* is 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,
```

```
    ...,
    sha384
}
```

This structure identifies a hash algorithm. The value `sha256` indicates SHA-256 as specified in 5.3.3. The value `sha384` indicates SHA-384 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,
    ...
    inlineP2pcdRequest      SequenceOfHashedId3 OPTIONAL,
    requestedCertificate    Certificate 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 does not know the entire certificate. A specification of this peer-to-peer certificate distribution (P2PCD) mechanism is given in Clause 8. This field is used for the out-of-band flavor of P2PCD and shall only be present if `inlineP2pcdRequest` is not present. The HashedId3 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.

— `missingCrlIdentifier`, if present, is used by the SDS to request CRLs which it knows to have been issued but has 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 6.3.33, 6.3.34, and 6.3.36. An `encryptionKey` field of type symmetric should only be used if the SignedData containing this field is securely encrypted by some means.

— `inlineP2pcdRequest`, if present, is used by the SDS to request unknown certificates per the inline peer-to-peer certificate distribution mechanism is given in Clause 8. This field shall only be present if `p2pcdLearningRequest` is not present. The HashedId3 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.

— `requestedCertificate`, if present, is used by the SDS to provide certificates per the "inline" version of the peer-to-peer certificate distribution mechanism given in Clause 8.

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 typeEccP256CurvePoint or EccP384CurvePoint), then the elliptic curve point is encoded in compressed form, i.e., such that the choice indicated within the Ecc*CurvePoint 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 1/10th 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 1/10th 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. The HashedId3 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.
— `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.

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.

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.

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)),
    uncompressedP256    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 least significant bit of y: if the least significant bit of y is 0, type takes the value `compressed-y-0`, and if the least significant bit 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 EccP384CurvePoint

```
EccP384CurvePoint::= CHOICE  {
    x-only              OCTET STRING (SIZE (48)),
    fill                NULL, -- consistency w 1363 / X9.62
    compressed-y-0      OCTET STRING (SIZE (48)),
    compressed-y-1      OCTET STRING (SIZE (48)),
    uncompressedP384    SEQUENCE  {
        x OCTET STRING (SIZE (48)),
        y OCTET STRING (SIZE (48))
    }
}
```

This structure specifies a point on an elliptic curve in Weierstrass form defined over a 384-bit prime number. The only supported such curve in this standard is Brainpool p384r1 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 48 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 least significant bit of y: if the least significant bit of y is 0, type takes the value `compressed-y-0`, and if the least significant bit 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 48 octets in network byte order.

6.3.25 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. The HashedId8 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.

  — 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 `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.

6.3.26 HashedId3

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

SequenceOfHashedId3 ::= SEQUENCE OF HashedId3
```

This data structure contains the truncated hash of another data structure. The HashedId3 for a given data structure is calculated by calculating the hash of the encoded data structure and taking the low-order three bytes of the hash output. If the data structure is subject to canonicalization it is canonicalized before hashing. The low-order three bytes are the last three bytes of the hash when represented in network byte order. See Example below.

The hash algorithm to be used to calculate a HashedId3 within a structure depends on the context. In this standard, for each structure that includes a HashedId3 field, the corresponding text indicates how the hash algorithm is determined.

Example: Consider the SHA-256 hash of the empty string:
`SHA-256("") =`
`e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b78`**`52b855`**

The HashedId3 of this hash was highlighted above and corresponds to the following:
`HashedId3 = 52b855.`

6.3.27 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 hash of the encoded data structure and taking the low-order eight bytes of the hash output. If the data structure is subject to canonicalization it is canonicalized before hashing. The low-order eight bytes are the last eight bytes of the hash when represented in network byte order. See Example below.

The hash algorithm to be used to calculate a HashedId8 within a structure depends on the context. In this standard, for each structure that includes a HashedId8 field, the corresponding text indicates how the hash algorithm is determined.

Example: Consider the SHA-256 hash of the empty string:
SHA-256("") =
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934c**a495991b7852b855**

The HashedId8 of this hash was highlighted above and corresponds to the following:
HashedId8 = a495991b7852b855.

6.3.28 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 hash of the encoded data structure and taking the low-order ten bytes of the hash output. If the data structure is subject to canonicalization it is canonicalized before hashing.The low-order ten bytes are the last ten bytes of the hash when represented in network byte order. See Example below.

The hash algorithm to be used to calculate a HashedId10 within a structure depends on the context. In this standard, for each structure that includes a HashedId10 field, the corresponding text indicates how the hash algorithm is determined.

Example: Consider the SHA-256 hash of the empty string:
SHA-256("") =
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b**934ca495991b7852b855**

The HashedId10 of this hash was highlighted above and corresponds to the following:
HashedId10 = 934ca495991b7852b855.

6.3.29 Signature

```
Signature ::= CHOICE  {
    ecdsaNistP256Signature        EcdsaP256Signature,
    ecdsaBrainpoolP256r1Signature EcdsaP256Signature,
    ...,
    ecdsaBrainpoolP384r1Signature EcdsaP384Signature,
}
```

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

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.

6.3.30 EcdsaP256Signature

```
EcdsaP256Signature ::= SEQUENCE  {
    rSig     EccP256CurvePoint,
```

```
    sSig      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.[12]

Encoding considerations: If this structure is encoded for hashing, the EccP256CurvePoint in `rSig` shall be taken to be of form `x-only`.

NOTE— When the signature is of form `x-only`, the x-value in `rSig` is an integer mod n, the order of the group; when the signature is of form `compressed-y-*`, the x-value in `rSig` is an integer mod p, the underlying prime defining the finite field. In principle this means that to convert a signature from form `compressed-y-*` to form `x-only`, the x-value should be checked to see if it lies between n and p and reduced mod n if so. In practice this check is unnecessary: Haase's Theorem states that difference between n and p is always less than $2\sqrt{p}$, and so the chance that an integer lies between n and p, for a 256-bit curve, is bounded above by approximately $\sqrt{p}/p$ or 2^{-128}. For the 256-bit curves in this standard, the exact values of n and p in hexadecimal are:

NISTp256:

— p = FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF

— n = FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

Brainpoolp256:

— p = A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377

— n = A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7

6.3.31 EcdsaP384Signature

```
EcdsaP384Signature ::= SEQUENCE  {
    rSig      EccP384CurvePoint,
    sSig      OCTET STRING (SIZE (48))
}
```

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 EccP384CurvePoint indicating the selection `x-only`.

[12] 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.

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 EccP384CurvePoint indicating the choice `compressed-y-0`, `compressed-y-1`, or `uncompressed` at the sender's discretion.[13]

Encoding considerations: If this structure is encoded for hashing, the EccP256CurvePoint in `rSig` shall be taken to be of form `x-only`.

NOTE—When the signature is of form `x-only`, the x-value in `rSig` is an integer mod n, the order of the group; when the signature is of form `compressed-y-*`, the x-value in `rSig` is an integer mod p, the underlying prime defining the finite field. In principle this means that to convert a signature from form `compressed-y-*` to form `x-only`, the x-value should be checked to see if it lies between n and p and reduced mod n if so. In practice this check is unnecessary: Haase's Theorem states that difference between n and p is always less than $2\sqrt{p}$, and so the chance that an integer lies between n and p, for a 384-bit curve, is bounded above by approximately $\sqrt{p}/p$ or 2^{-192}. For the 384-bit curve in this standard, the exact values of n and p in hexadecimal are:

— p = 8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901D1A71874700133107EC53

— n = 8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF3AB6AF6B7FC3103B883202E9046565

6.3.32 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 `RecipientInfo`s, 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.

[13] 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.

6.3.33 RecipientInfo

```
RecipientInfo ::= CHOICE  {
    pskRecipInfo        PreSharedKeyRecipientInfo,
    symmRecipInfo       SymmRecipientInfo,
    certRecipInfo       PKRecipientInfo,
    signedDataRecipInfo PKRecipientInfo,
    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 a public key encryption scheme, where the public encryption key was obtained from a 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 a public encryption key, where the encryption key was obtained as the public response encryption key from a SignedData. In this case, the parameter P1 to ECIES as defined in 5.3.5 is the SHA-256 hash of the Ieee1609Dot2-Data containing the response encryption key.

— `rekRecipInfo`: The data encryption key was encrypted using a public key that was not obtained from a SignedData. In this case, the parameter P1 to ECIES as defined in 5.3.5 is the hash of the empty string.

See Annex C.7 for guidance on when it may be appropriate to use each of these approaches.

6.3.34 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 COER encoding of a SymmetricEncryptionKey structure containing the symmetric key in question. The symmetric key may be established by any appropriate means agreed by the two parties to the exchange.

6.3.35 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. It consists of the low-order 8 bytes of the SHA-256 hash of the COER

encoding of a SymmetricEncryptionKey structure containing the symmetric key in question. The symmetric key may be established by any appropriate means agreed by the two parties to the exchange.

— `encKey` contains the encrypted data encryption key within an AES-CCM ciphertext.

6.3.36 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. Specifically, depending on the choice indicated by the containing RecipientInfo structure:

  – If the containing `RecipientInfo` structure indicates `certRecipInfo`, this field contains the HashedId8 of the certificate. The HashedId8 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.

  – If the containing `RecipientInfo` structure indicates `signedDataRecipInfo`, this field contains the HashedId8 of the Ieee1609Dot2Data of type `signed` that contained the encryption key, with that Ieee1609Dot2Data canonicalized per 6.3.4. The HashedId8 is calculated with SHA-256.

  – If the containing `RecipientInfo` structure indicates `rekRecipInfo`, this field contains the HashedId8 of the COER encoding of a PublicEncryptionKey structure containing the response encryption key. The HashedId8 is calculated with SHA-256.

— `encKey` contains the encrypted key.

6.3.37 EncryptedDataEncryptionKey

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

This data structure contains an encrypted data encryption key.

Critical information fields: If present and applicable to the receiving SDEE, this is a critical information field as defined in 5.2.5. If an implementation receives an encrypted SPDU and determines that one or more RecipientInfo fields are relevant to it, and if all of those RecipientInfos contain an EncryptedDataEncryptionKey such that the implementation does not recognize the indicated CHOICE, the implementation shall indicate that the encrypted SPDU is not decryptable.

6.3.38 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.

6.3.39 SymmetricCiphertext

```
SymmetricCiphertext ::= CHOICE  {
    aes128ccm           Aes128CcmCiphertext,
    ...
}
```

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.

6.3.40 Aes128CcmCiphertext

```
Aes128CcmCiphertext ::= 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.

6.3.41 Countersignature

```
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
          })
        })
      })
    })
  })
```

This data structure is used to perform a countersignature over an already-signed SPDU. This is the profile of an Ieee1609Dot2Data containing a signedData. The tbsData within content is comprised of a payload containing the hash (extDataHash) of the externally generated, pre-signed SPDU over which the countersignature is performed.

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 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 or EccP384CurvePoint indicating the choice `x-only`.

Whole-certificate hash: If the entirety of a certificate is hashed to calculate a HashedId3, HashedId8, or HashedId10, the algorithm used for this purpose is known as the *whole-certificate hash*.

— The whole-certificate hash is SHA-256 if the certificate is an implicit certificate.

— The whole-certificate hash is SHA-256 if the certificate is an explicit certificate and toBeSigned.-verifyKeyIndicator.verificationKey is a EccP256CurvePoint .

— The whole-certificate hash is SHA-384 if the certificate is an explicit certificate and toBeSigned.-verifyKeyIndicator.verificationKey is a EccP384CurvePoint .

6.4.4 CertificateType

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

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

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.

6.4.5 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,
    ...,
    sha384AndDigest         HashedId8
}
```

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` or `sha384AndDigest`:

— The structure contains the HashedId8 of the issuing certificate, where the certificate is canonicalized as specified in 6.4.3 before hashing and the HashedId8 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.

— The hash algorithm to be used to generate the hash of the certificate for verification is SHA-256 (in the case of `sha256AndDigest`) or SHA-384 (in the case of `sha384AndDigest`).

— 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 theToBeSignedCertificate.

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.

6.4.8 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. The HashedId3 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.

— `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 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 ECC curve point, 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 Hash (*Data input*) || Hash (*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 (canonicalized entirety of issuer Certificate)]. See 5.3.2 for further discussion, including material differences between this standard and SEC4 regarding how the hash function output is converted from a bit string to an integer.

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 signed 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 signed 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 signed 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 and 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 years.

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 signed 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 SPDU shall indicate that the signed SPDU 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 recognize RegionAndSubregions or CountryAndSubregions values 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.

Critical information fields: RegionAndSubregions is a critical information field as defined in 5.2.5. An implementation that does not detect or recognize the the region or subregions values when verifying a signed SPDU shall indicate that the signed SPDU is invalid.

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.

Consistency with signed SPDU. As noted in 5.1.1, consistency between the SSP and the signed SPDU is defined by rules specific to the given PSID and is out of scope for this standard.

Consistency with issuing certificate.

If a certificate has an `appPermissions` entry *A* for which the ssp field is omitted, *A* is consistent with the issuing certificate if the issuing certificate contains a PsidSspRange *P* for which the following holds:

— The `psid` field in *P* is equal to the `psid` field in *A* and one of the following is true:

  — The `sspRange` field in *P* indicates `all`.

  — The `sspRange` field in *P* indicates `opaque` and one of the entries in `opaque` is an OCTET STRING of length 0.

For consistency rules for other forms of the `ssp` field, see the following subclauses.

6.4.29 ServiceSpecificPermissions

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

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. SSPs may be PSID-specific octet strings or bitmap-based. See Annex C for further discussion of how application specifiers may choose which SSP form to use.

Consistency with issuing certificate.

If a certificate has an `appPermissions` entry *A* for which the ssp field is `opaque`, *A* is consistent with the issuing certificate if the issuing certificate contains one of the following:

— (OPTION 1) A SubjectPermissions field indicating the choice `all` and no PsidSspRange field containing the `psid` field in *A*;

— (OPTION 2) A PsidSspRange P for which the following holds:

  — The `psid` field in *P* is equal to the `psid` field in *A* and one of the following is true:

    — The `sspRange` field in *P* indicates `all`.

    — The `sspRange` field in *P* indicates `opaque` and one of the entries in the `opaque` field in *P* is an OCTET STRING identical to the `opaque` field in *A*.

For consistency rules for other types of `ServiceSpecificPermissions`, see the following subclauses.

6.4.30 BitmapSsp

```
BitmapSsp ::= OCTET STRING (SIZE(0..31))
```

This structure represents a bitmap representation of a SSP. The mapping of the bits of the bitmap to constraints on the signed SPDU is PSID-specific.

Consistency with issuing certificate.

If a certificate has an `appPermissions` entry *A* for which the `ssp` field is `bitmapSsp`, *A* is consistent with the issuing certificate if the issuing certificate contains one of the following:

— (OPTION 1) A SubjectPermissions field indicating the choice `all` and no PsidSspRange field containing the `psid` field in *A*;

— (OPTION 2) A PsidSspRange *P* for which the following holds:

  — The `psid` field in *P* is equal to the `psid` field in *A* and one of the following is true:

— EITHER The sspRange field in *P* indicates all

— OR The sspRange field in *R* indicates bitmapSspRange and for every bit set to 1 in the sspBitmask in *R*, the bit in the identical position in the sspValue in *P* is set equal to the bit in that position in the sspValue in *R*.

NOTE—A BitmapSsp *B* is consistent with a BitmapSspRange *R* if for every bit set to 1 in the sspBitmask in *R*, the bit in the identical position in *B* is set equal to the bit in that position in the sspValue in *R*. For each bit set to 0 in the sspBitmask in *R*, the corresponding bit in the identical position in *B* may be freely set to 0 or 1, i.e. if a bit is set to 0 in the sspBitmask in *R*, the value of corresponding bit in the identical position in *B* has no bearing on whether *B* and *R* are consistent.

6.4.31 PsidGroupPermissions

```
PsidGroupPermissions ::= SEQUENCE  {
    subjectPermissions SubjectPermissions,
    minChainLength     INTEGER DEFAULT 1,
    chainLengthRange   INTEGER DEFAULT 0,
    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.

— minChainLength and chainLengthRange indicate how long the certificate chain from this certificate to the end-entity certificate is permitted to be. As specified in 5.1.2.1, the length of the certificate chain is the number of certificates "below" this certificate in the chain, down to and including the end-entity certificate. The length is permitted to be (a) greater than or equal to minChainLength certificates and (b) less than or equal to minChainLength + chainLengthRange certificates. A value of 0 for minChainLength is not permitted when this type appears in the certIssuePermissions field of a ToBeSignedCertificate; a certificate that has a value of 0 for this field is invalid. The value −1 for chainLengthRange is a special case: if the value of chainLengthRange is −1 it indicates that the certificate chain may be any length equal to or greater than minChainLength. See the examples below for further discussion.

— eeType takes one or more of the values app and enroll 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 is allowed to end in an authorization certificate, i.e., a certfficate in which these permissions appear in an appPermissions field (in other words, if the field does not indicate app but the chain ends in an authorization certificate, the chain shall be considered invalid). If this field indicates enroll, the chain is allowed to end in an enrollment certificate, i.e., a certificate in which these permissions appear in a certReqPermissions permissions field), or both (in other words, if the field does not indicate app but the chain ends in an authorization certificate, the chain shall be considered invalid). Different instances of PsidGroupPermissions within a ToBeSignedCertificate may have different values for eeType.

For examples, see Annexes D.5.3 and D.5.4.

6.4.32 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.

6.4.33 EndEntityType

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

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

6.4.34 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 issue or request certificates for any SSP for that PSID.

6.4.35 SspRange

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

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

Consistency with issuing certificate.

If a certificate has a PsidSspRange *A* for which the ssp field is `opaque`, *A* is consistent with the issuing certificate if the issuing certificate contains one of the following:

— (OPTION 1) A SubjectPermissions field indicating the choice `all` and no PsidSspRange field containing the `psid` field in *A*;

— (OPTION 2) a PsidSspRange P for which the following holds:

  — The `psid` field in *P* is equal to the `psid` field in *A* and one of the following is true:

    — The `sspRange` field in *P* indicates `all`.

    — The `sspRange` field in *P* indicates `opaque`, and the `sspRange` field in *A* indicates `opaque`, and every OCTET STRING within the `opaque` in *A* is a duplicate of an OCTET STRING within the `opaque` in *P*.

If a certificate has a PsidSspRange *A* for which the ssp field is `all`, *A* is consistent with the issuing certificate if the issuing certificate contains a PsidSspRange *P* for which the following holds:

— (OPTION 1) A SubjectPermissions field indicating the choice `all` and no PsidSspRange field containing the `psid` field in *A*;

— (OPTION 2) A PsidSspRange *P* for which the `psid` field in *P* is equal to the `psid` field in *A* and the `sspRange` field in *P* indicates `all`.

For consistency rules for other types of SspRange, see the following subclauses.

NOTE— The choice "`all`" may also be indicated by omitting the SspRange in the enclosing PsidSspRange structure. Omitting the SspRange is preferred to explicitly indicating "`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.

6.4.36 BitmapSspRange

```
BitmapSspRange ::= SEQUENCE  {
    sspValue            OCTET STRING (SIZE(1..32)),
    sspBitmask          OCTET STRING (SIZE(1..32)),
}
```

This structure represents a bitmap representation of a SSP. The `sspValue` indicates permissions. The `sspBitmask` contains an octet string used to permit or constrain `sspValue` fields in issued certificates. The `sspValue` and `sspBitmask` fields shall be of the same length.

Consistency with issuing certificate.

If a certificate has an `PsidSspRange` value *P* for which the `sspRange` field is `bitmapSspRange`, *P* is consistent with the issuing certificate if the issuing certificate contains one of the following:

— (OPTION 1) A SubjectPermissions field indicating the choice `all` and no PsidSspRange field containing the `psid` field in *P*;

— (OPTION 2) A PsidSspRange *R* for which the following holds:

  — The `psid` field in *R* is equal to the `psid` field in *P* and one of the following is true:

    — EITHER The `sspRange` field in *R* indicates `all`.

    — OR The `sspRange` field in *R* indicates `bitmapSspRange` and for every bit set to 1 in the `sspBitmask` in *R*:

      — The bit in the identical position in the `sspBitmask` in *P* is set equal to 1, AND

      — The bit in the identical position in the `sspValue` in *P* is set equal to the bit in that position in the `sspValue` in *R*.

Reference ETSI TS 103 097 [B7] for more information on bitmask SSPs.

6.4.37 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.

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.

6.4.38 PublicVerificationKey

```
PublicVerificationKey ::= CHOICE  {
    ecdsaNistP256              EccP256CurvePoint,
    ecdsaBrainpoolP256r1       EccP256CurvePoint,
    ... ,
    ecdsaBrainpoolP384r1       EccP384CurvePoint
}
```

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 or EccP384CurvePoint 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 17.

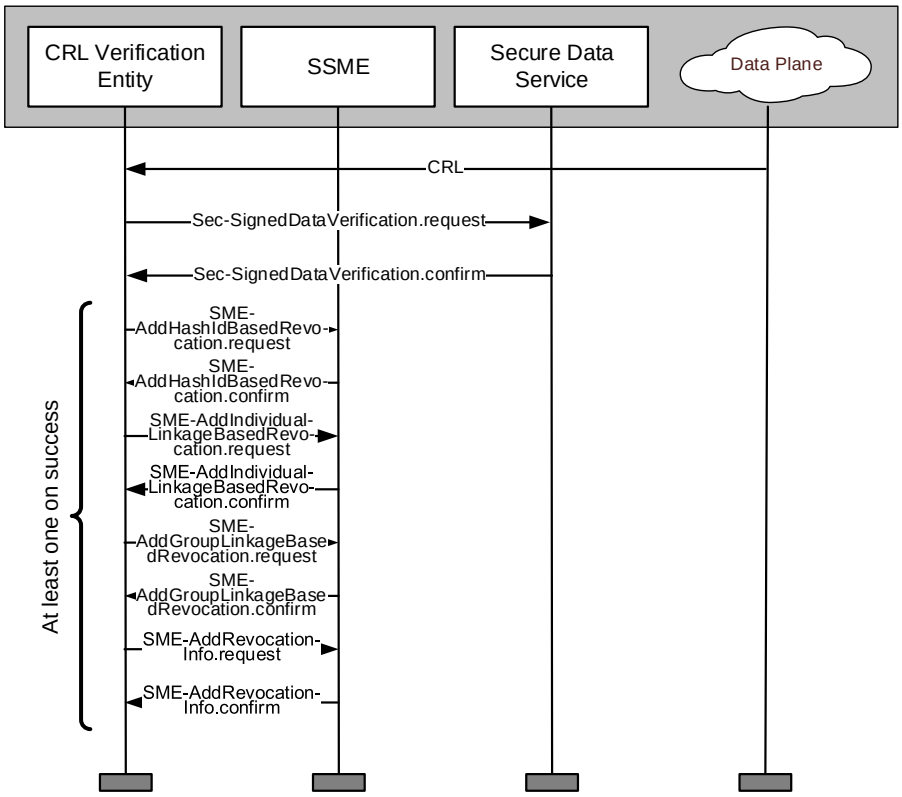


Figure 17—CRL Verification Entity

7.3 Data structures

7.3.1 General

Subclause 7.3 specifies the CRL contents using ASN.1. Subclause B.3 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,
    crlCraca            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.
- `crlCraca` 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 `crlCraca` is consistent with the `associatedCraca` field in the Service Specific Permissions as defined in 7.4.3.3. The HashedId8 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.
- `issueDate` specifies the time when the CRL was issued.
- `nextCrl` contains the time when the next CRL with the same `crlSeries` and `crlCraca` 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 (`crlCraca`, `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.

NOTE—This mechanism is for future use; details are not specified in this version of the standard.

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 `crlCraca` 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. The HashedId10 is calculated with the whole-certificate hash algorithm, determined as described in 6.4.3.

— `expiry` is the value computed from the validity period's start and duration values in that certificate.

7.3.6 ToBeSignedLinkageValueCrl

```
ToBeSignedLinkageValueCrl ::= SEQUENCE  {
    iRev                IValue,
    indexWithinI        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 `indvidual` or `groups`.

— `indexWithinI` is a counter that is set to 0 for the first CRL issued for the indicated combination of `crlCraca`, `crlSeries`, and `iRev`, and increments by 1 every time a new full or delta CRL is issued for the indicated `crlCraca` 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 (`crlCraca`, `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 SignedDataPayload 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 11, a valid signed CRL also meets one of the following conditions:

— The CRL was signed by the CRACA indicated by the `crlCraca`, or

— The CRL was signed by a certificate which was issued by the CRACA indicated by the `crlCraca`.

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 `crlCraca` field in the CrlContents structure.

— If this takes the value `isCraca`, the `crlCraca` 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
Security Profile Version	IEEE Std 1609.2a-2017	
Name	"IEEE 1609.2 security profile for Certificate Revocation List"	
PSIDs	The value indicated in IEEE Std 1609.12 for "Certificate Revocation List Application"	
Other considerations		

7.4.4.2 Sending

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

Field	Value	Notes
Sign Data	True	All CRLs are signed
Signed Data in Payload	True	CRL is contained within the encapsulating SignedData
External Data	False	
External Data Source	N/a	
External Data Hash Algorithm	N/a	
Set Generation Time in Security Headers	False	Included in CRL payload
Set Generation Location in Security Headers	False	Not used
Set Expiry Time in Security Headers	False	Included in CRL payload
Signed SPDU Lifetime	N/a	Not used as expiry is not used
Signer Type Self	Prohibited	CRLs are signed with a certificate
Signer Type Self Permitted	N/a	Signer type self is prohibited
Verification Key Location for Signer Type Self	N/a	Signer type self is prohibited
Signer Identifier Policy Type	Simple	Signer type self is prohibited
Simple Signer Identifier Policy: Minimum InterCert Time	All	
Simple Signer Identifier Policy: Exceptions	None	
Simple Signer Identifier Policy: Signer Identifier Cert Chain Length	1	1 is enough to get back to the CRACA
Text Signer Identifier Policy	N/a	
Sign With Fast Verification	True	Allow fast verification
EC Point Format	Compressed	Reduce key size
p2pcd_flavor	None	Full cert chain is attached
p2pcd_maxResponseBackoff	N/a	
p2pcd_responseActiveTimeout	N/a	
p2pcd_requestActiveTimeout	N/a	
p2pcd_observedRequestTimeout	N/a	
p2pcd_currentlyUsedTriggerCertificateTime	N/a	
p2pcd_responseCountThreshold	N/a	
Repeat Signed SPDUs	N/a	CRL transmission is different from the CRL signing application, so this field doesn't apply
Time Between Signing	N/a	CRL transmission is different from the CRL signing application, so this field doesn't apply
Encrypt Data	False	

7.4.4.3 Receiving

Field	Value	Notes
Use Preprocessing	False	CRLs have the full certificate chain and do not use P2PCD, so no need for preprocessing
Verify Data	True	
Maximum Full Certificate Chain Length	8	
Relevance: Replay	False	Replayed CRLs are not an attack
Relevance: Generation Time in Past	False	Generation time of CRL is not used to decide whether or not it is relevant
Validity Period	N/a	
Relevance: Generation Time in Future	False	CRLs could conceivably be issued to take effect in the future
Acceptable Future Data Period	N/a	
Generation Time Source	Payload	Generation time is compared with CRL signing cert validity period
Relevance: Expiry Time	N/a	CRLs do not expire as such—even if another CRL is issued, the previous CRL is still valid for use
Expiry Time Source	Payload	
Consistency: Generation Location	False	Generation location is irrelevant for CRLs
Relevance: Generation Location Distance	N/a	
Validity Distance	N/a	
Generation Location Source	N/a	
Additional Geographic Consistency Conditions	False	CRL does not carry any geographic information.
Identified Region Representation Accuracy	N/A	CRL does not require location validity checks
Overdue CRL Tolerance	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
Relevance: Certificate Expiry	True	A CRL signed with an expired certificate should not be accepted
Encrypted Data	False	CRLs are not encrypted

7.4.4.4 Security management

Field	Value	Notes
Signing Key Algorithm	ecdsaNistP256, ecdsaBrainpoolP256r1, ecdsaBrainpoolP384r1	
Encryption Algorithm	n/a	
Implicit or Explicit Certificates	Implicit	
EC Point Format	Compressed	
Supported Geographic Regions	None	CRLs are not limited by geographic region
Maximum Full Certificate - Chain Length	7	There may be 8 certificates in total in the chain (and 7 inter-certificate gaps).
Use Individual Linkage ID	False	CRL signers use identified certs and are revoked by hash if necessary
Use Group Linkage ID	False	CRL signers use identified certs and are revoked by hash if necessary
Signature Algorithms in Chain or CRL	ecdsaNistP256, ecdsaBrainpoolP256r1, ecdsaBrainpoolP384r1	May be constrained by the security profile for the relevant application

7.4.4.5 Other

Field	Value	Notes
Fields that may be subject to policy update	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(256)

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 SDEE 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 WAVE Security Services instance that received the trigger SPDU uses P2PCD learning requests to request peer instances 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.

The IEEE 1609.2 security profile (see Annex C) provides a means for SDEE specifies to specify whether P2PCD is used by an SDEE, and if so what flavor is used and what parameters are provided.

8.2 P2PCD operations

8.2.1 General

The following is an overview of P2PCD operations.

There are two "flavors" of P2PCD, "inline" and "out-of-band". In inline P2PCD, the certificates are included directly in signed SPDUs from the trigger SDEE; in out-of-band P2PCD, the certificates are transmitted in separate PDUs. In both flavors:

— Signed SPDUs are received by a *trigger SDEE* and processed by the SDS. In the course of this processing:

  — If the signed SPDU 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 SPDU 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 it has not recently requested the sending of 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.

The differences between the inline and out-of-band approaches are:

— Request:

- In the out-of-band approach, P2PCD only supports requesting CA certificates. In the inline approach, the P2PCD request process is also triggered if the signed SPDU has a SignerInfo of type `digest` and the end-entity certificate indicated by this SignerInfo is unknown to the local SDEE. In other words, the inline approach can be used to request end-entity certificates.

- Response:
  - In the out-of-band P2PCD response process, the P2PCDE is requested by the SDS 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 P2PCDE instances 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. Out-of-band responses are specified in 8.2.4.2.2.
  - In the inline P2PCD response process, the SDS inserts P2PCD learning responses into the next SPDU sent by the SDEE. 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. Inline responses are specified in 8.2.4.2.3.

An example of information flows to support out-of-band P2PCD is given in the illustrative Figure 18. 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 20.

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. Responders stop sending responses once they have reached a prescribed configurable threshold number of responses as specified in the SDEE specification, for example in the IEEE 1609.2 security profile for that SDEE.

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.

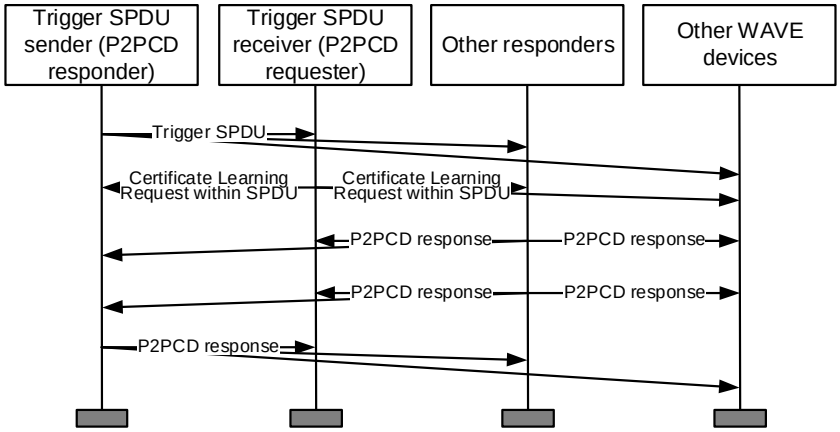


Figure 18—Overview of information flows for P2PCD

8.2.2 Functional entities

Figure 19 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 SSME via the SSME-Sec-SAP to enable it to determine whether to include P2PCD learning requests in SPDUs.
  - The SDS includes P2PCD learning requests in SPDUs when so requested by the SSME via the SSME-Sec-SAP.
  - In the inline case, the SDS includes requested certificates in SPDUs when so requested by the SSME via the SSME-Sec-SAP.
- 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.
  - In the inline case, the SSME requests the SDS via the SSME-Sec-SAP to include requested certificates in SPDUs.

— In the out-of-band case, 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 is only active in the out-of-band case. It 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 20 shows information flows for an instance of P2PCD, showing the information flows within instances of WAVE Security Services as well as between WAVE devices.

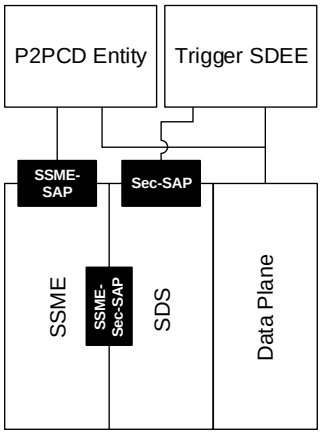


Figure 19—Functional entities involved in peer-to-peer certificate distribution

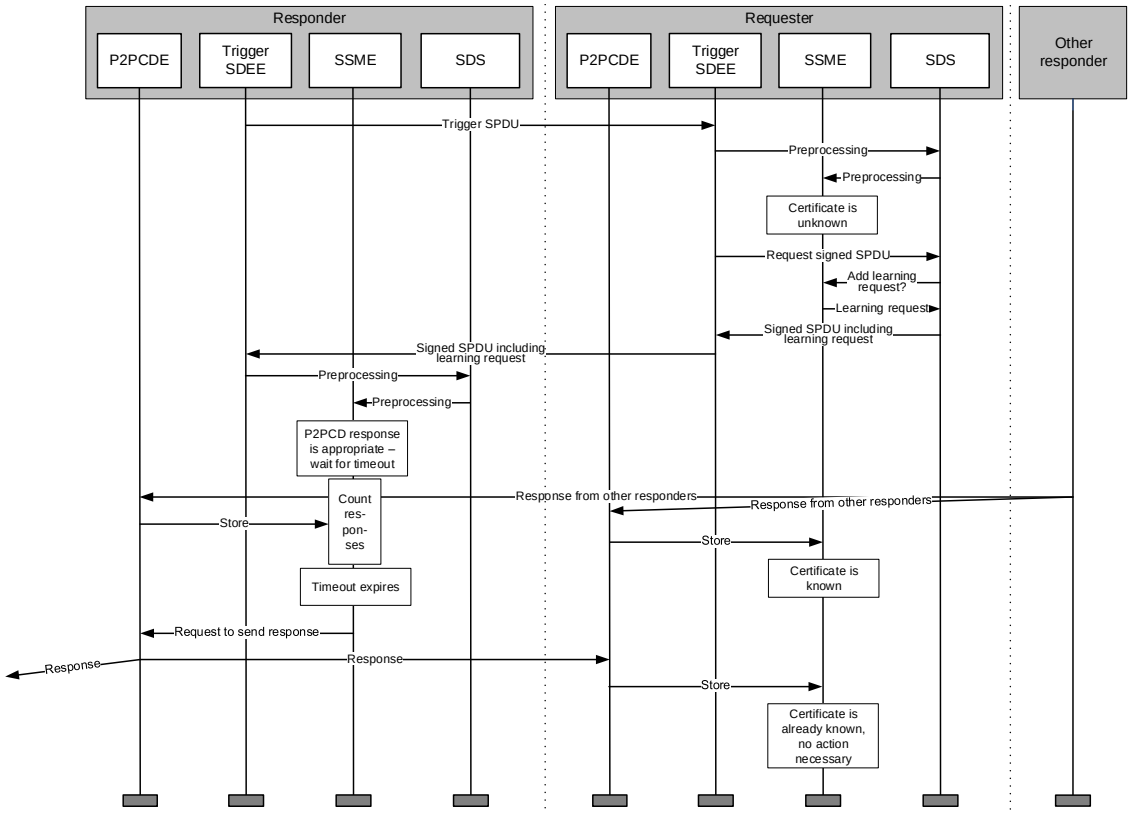


Figure 20—Overview of information flows showing functional entities

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_flavor` (SDEE ID *s*): An enumerated value taking the value “inline”, “Out of Band”, or “none” indicating which flavor of P2PCD is in use for the indicated SDEE.

The following parameters are used only if `p2pcd_flavor`(*s*) is “out of band”:

— `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`. This may take the value “0”, indicating that there is no restriction on including the same request in consecutive SPDUs.

— `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`. This may take the value “0”, indicating that there is no restriction on including a request even if the same request has recently been observed in a received SPDU.

— `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*. This may take the value "0", indicating that unless other exception conditions are met the SSME will send the response at the first opportunity.

— `p2pcd_responseActiveTimeout` (SDEE ID *s*): After the SSME triggers the response process in response to a certifiate request received via *s*, it does not trigger another response process until a time equal to `p2pcd_responseActiveTimeout` has passed. This may take the value "0", indicating that responses may be triggered whenever a request is received via *s* whether or not another request has recently been received via *s*.

— `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. This is only used in the out-of-band case.

— `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

8.2.4.1.1 Out of band

This subclause specifies requester role operations for the out-of-band flavor of 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, i.e., a query of SSME-CertificateInfo.request results in a *Result Code* from SSME-CertificateInfo.confirm of "certificate not found", 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 21. Requester behavior when multiple certificates may be the subject of a request is illustrated in Figure 22. Subclause D.4 provides additional figures breaking down the activities by functional elements and identifying information flows between them.

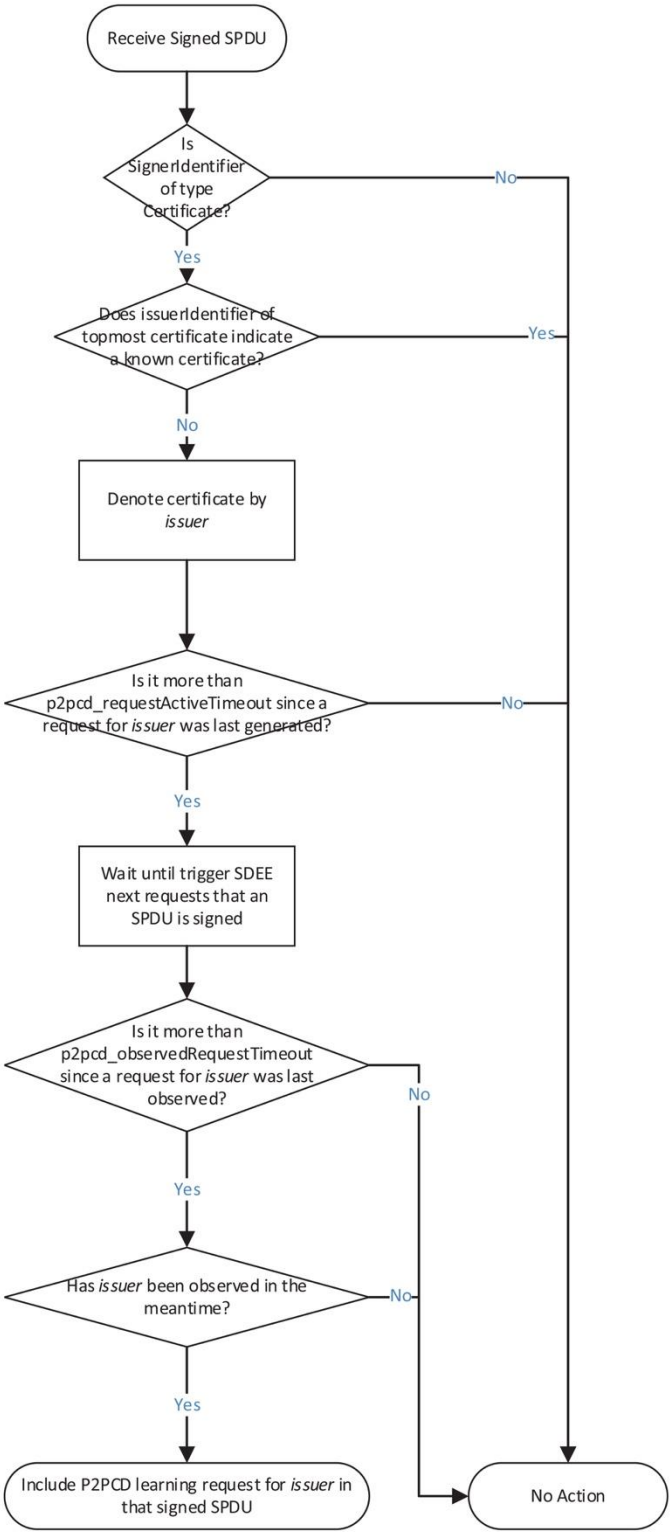


Figure 21—P2PCD requester behavior within WAVE Security Services

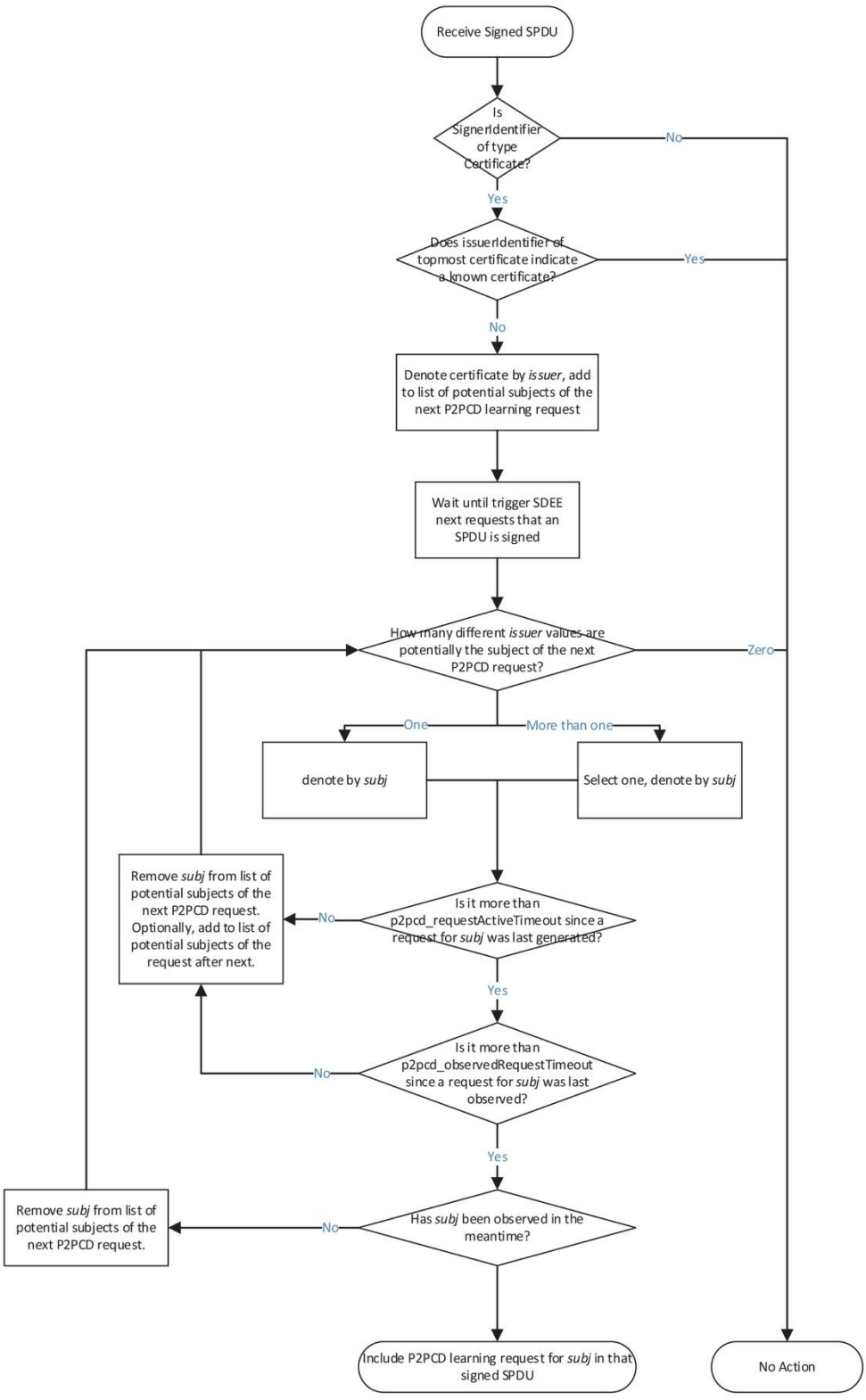


Figure 22—P2PCD requester behavior when multiple certificates may be the subject of a request

8.2.4.1.2 Inline

This subclause specifies requester role operations for the inline flavor of P2PCD for a single SDEE.

a) The SDS maintains an internal array, *p2pcd_inline_potentiallyRequestedCerts*(*s*) of certificates that might be the subject of a request by that SDEE. The array *p2pcd_inline_potentiallyRequestedCerts*(*s*) is initialized to an empty array and set equal to the empty array every time the trigger SDEE requests (via Sec-SignedData.request) the generation of a signed SPDU.

b) When a trigger SDEE requests (via Sec-SecureDataPreprocessing.request) that the SDS preprocesses a signed SPDU *sp*:

   1) If the SignerIdentifier field in *sp* indicates type `digest`:

      i) If the digest is of a certificate that is not known to the SSME, i.e. a query of SSME-CertificateInfo.request results in a *Result Code* from SSME-CertificateInfo.confirm of "Certificate not found", the SDS calculates the HashedId3 derived from `digest` and adds it to *p2pcd_inline_potentiallyRequestedCerts*(*s*).

   2) If the SignerIdentifier field in *sp* indicates type `certificate`:

      i) If the `issuer` field in the highest certificate in the chain contained in the SignerIdentifier indicates a certificate that is not known to the SSME, i.e., a query of SSME-CertificateInfo.request with that field results in a *Result Code* from SSME-Certificate-Info.confirm of "Certificate not found", then the SDS calculates the HashedId3 derived from `issuer` and adds it to *p2pcd_inline_potentiallyRequestedCerts*(*s*).

c) When a trigger SDEE requests (via Sec-SignedDataVerification.request) that the SDS verifies a signed SPDU *sp*:

   1) If Sec-SignedDataVerification.confirm returns the field *Unrecognized Id*, the SDS adds the HashedId3 derived from the *Unrecognized Id* to *p2pcd_inline_potentiallyRequestedCerts*(*s*).

d) When the SDS is requested (via Sec-SignedData.request) to sign an SPDU on behalf of SDEE *s*:

   1) If *p2pcd_inline_potentiallyRequestedCerts*(*s*) is not empty, the SDS selects one or more of the entries in *p2pcd_inline_potentiallyRequestedCerts*(*s*) for inclusion in the signed SPDU. The criteria for selection and the number of entries selected may be implementation-specific. The entries are included in the `inlineP2pcdRequest` field.

   2) The SDS sets the array *p2pcd_inline_potentiallyRequestedCerts*(*s*) to the empty array.

8.2.4.2 Responder role

8.2.4.2.1 General

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.

8.2.4.2.2 Out of band

If the P2PCD out-of-band flavor 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.

   1) 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 less 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.

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 23. Subclause D.4 provides additional figures breaking down the activities by functional elements and identifying information flows between them.

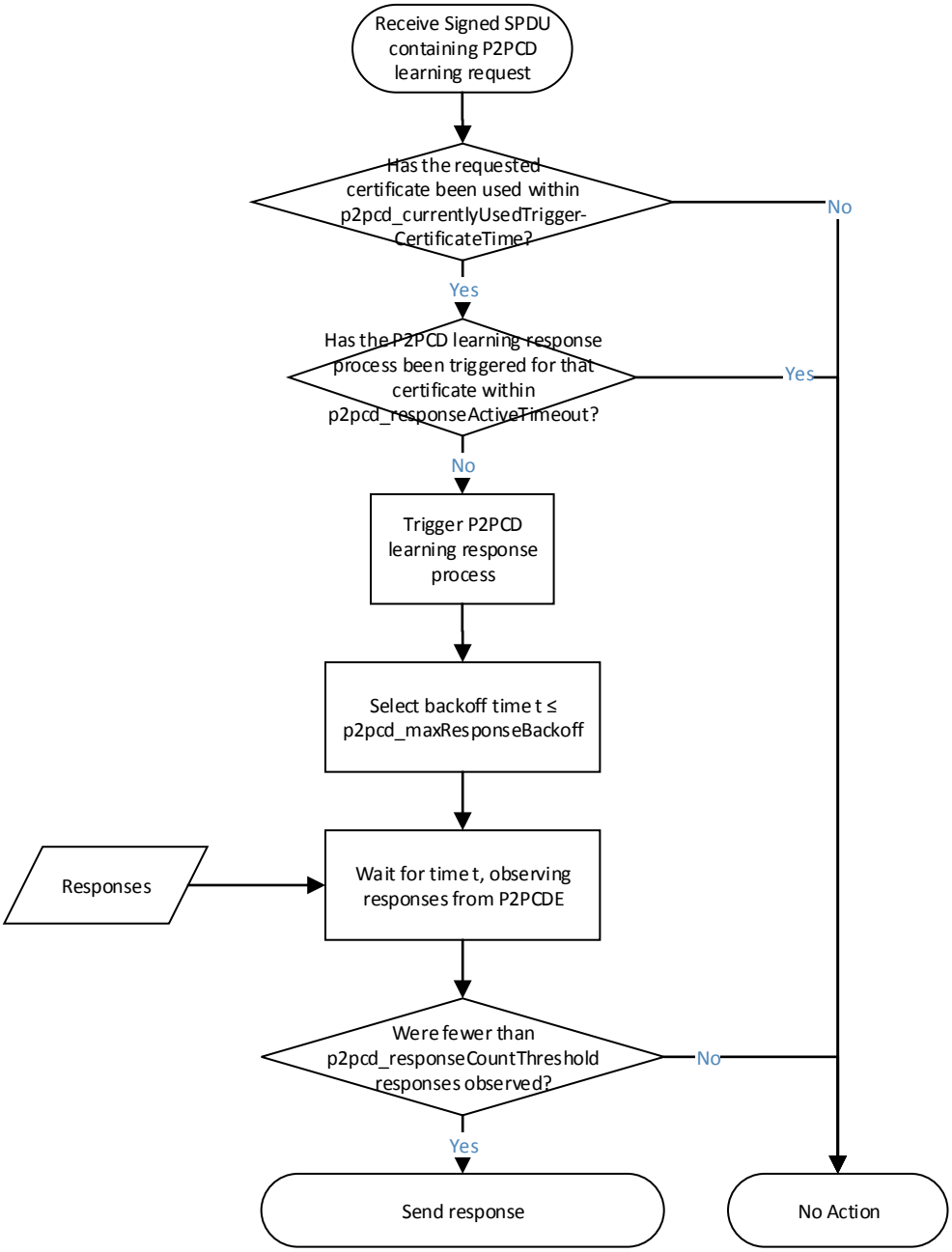


Figure 23—Interactive form of P2PCD responder behavior within WAVE Security Services

8.2.4.2.3 Inline

If the P2PCD inline flavor is in use, the P2PCD learning request may contain more than one entry. The request consists of all the entries in the `p2pcdLearningRequest` field and the `additionalP2pcdRequest` field in the HeaderInfo of a recently received signed SPDU. Operations proceed as follows:

a) The SDS maintains an array, *p2pcd_inline_requestedCerts(s)*, of certificates that have been requested by the SDS supporting a particular SDEE *s*.

b) The process starts with *p2pcd_inline_requestedCerts(s)* set equal to the empty array.

c) When the trigger SDEE requests (via Sec-SecureDataPreprocessing.request) that the SDS preprocesses a signed SPDU containing a P2PCD learning request, all the P2PCD learning requests from the signed SPDU are added to the array *p2pcd_inline_requestedCerts(s)*.

d) When the trigger SDEE requests (via Sec-SecureDataPreprocessing.request) that the SDS preprocesses a signed SPDU containing a `requestedCertificate` field:

   1) The SDS determines whether the HashedId3 of the certificate in the `requestedCertificate` field corresponds to any of the entries in *p2pcd_inline_requestedCerts(s)*. If this is the case, the SDS removes that entry from *p2pcd_inline_requestedCerts(s)*.

e) When SDS is requested (via Sec-SignedData.request) to sign an SPDU on behalf of SDEE *s*:

   1) If *p2pcd_inline_requestedCerts(s)* contains an indicator of the certificate that was used by the SDS to sign the most recent SPDU, then if the SDS creates a signed SPDU with the same certificate, it uses a SignerIdentifier indicating the choice `certificate` and containing the signing certificate.

   2) If *p2pcd_inline_requestedCerts(s)* does not contain an indicator of the certificate that was used by the SDS to sign the most recent SPDU, but does contain an indicator of a CA certificate known to the SDS (i.e. a query of SSME-CertificateInfo.request with that field results in a *Result Code* from SSME-CertificateInfo.confirm other than "Certificate not found" and that certificate has non-empty `certIssuePermissions` field), then the SDS selects one of those CA certificates and includes it in the `requestedCertificate` field of the signed SPDU.

f) The SDS sets *p2pcd_inline_requestedCerts*(*s*) to the empty array.

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) major-version-2(2)}

--
**************************************************************************
--
-- 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) major-version-2(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) major-version-2(2)}
;

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 and 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 24 illustrates the CMH model.

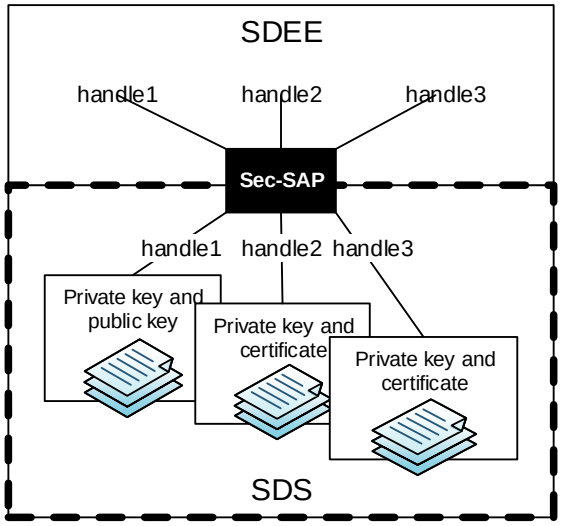


Figure 24—SDEE and Cryptomaterial Handle

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 25 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.

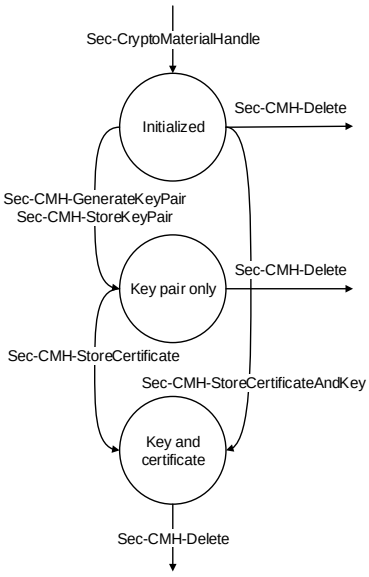


Figure 25—State diagram for Cryptomaterial Handles

9.2.2.3 Initialization

A CMH is created in the *Initialized* state as shown in Figure 26 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.

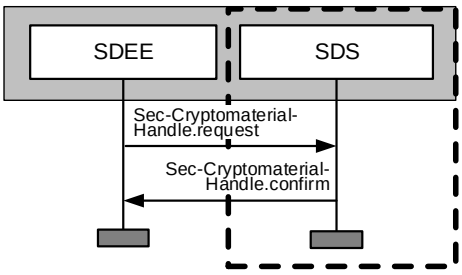


Figure 26—Process flow for obtaining a new Cryptomaterial Handle

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 27.

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 28.

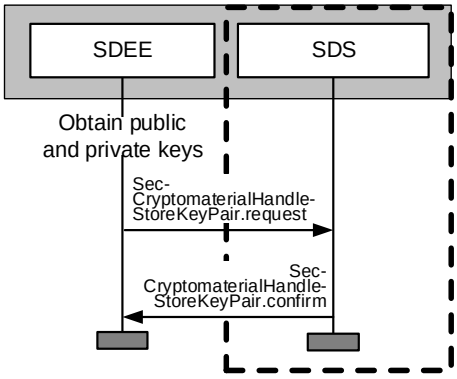


Figure 27—Transitioning a CMH from *Initialized* to *Key Pair Only* state with externally generated keys

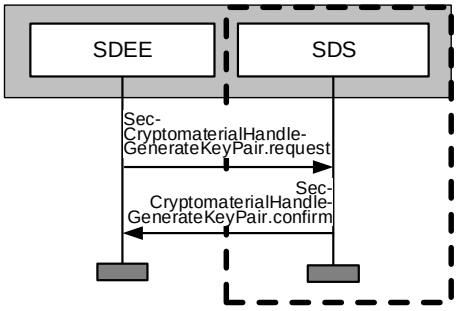


Figure 28—Transitioning a CMH from *Initialized* to *Key Pair Only* state with keys generated by the SDS

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 confirms 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 29.

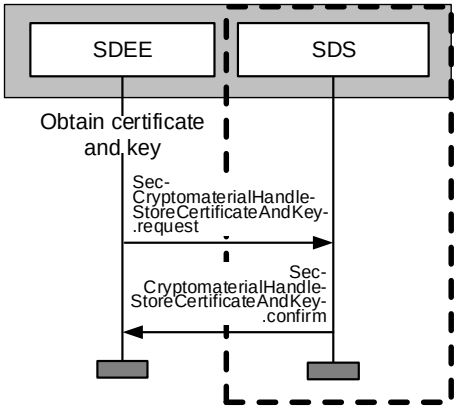


Figure 29—Transitioning a CMH directly from *Initialized* to *Key and Certificate* state

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 30.

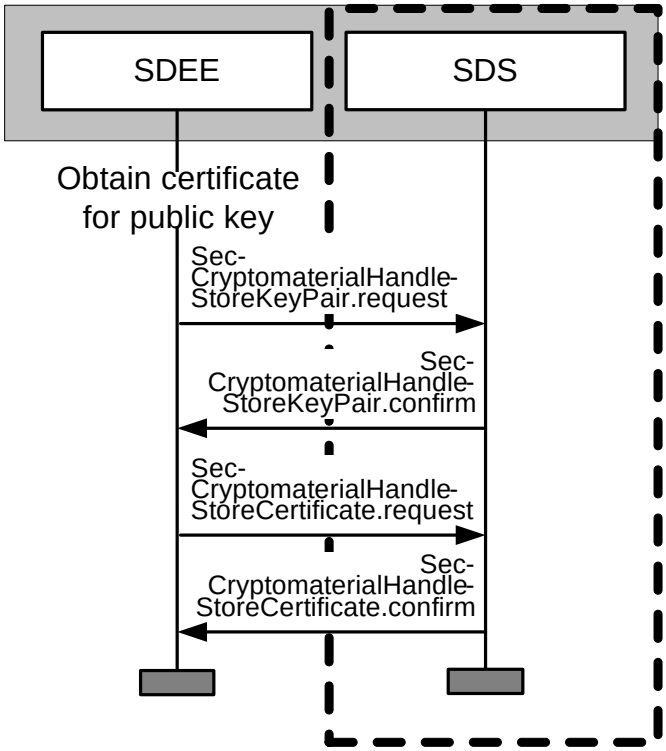


Figure 30—Transitioning a CMH from *Initialized* to *Key and Certificate* state via *Key Pair Only* state with externally generated keys

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 beginning with 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-Store-Certificate.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 31.

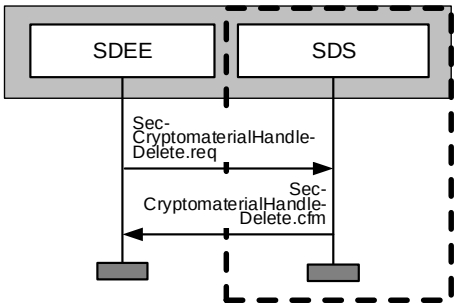


Figure 31—Process flow for deleting a Cryptomaterial Handle

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 32 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.

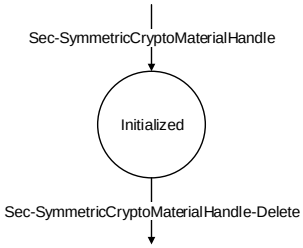


Figure 32—State diagram for Symmetric Cryptomaterial Handles

9.2.3.3 Initialization

A SCMH is created in the *Initialized* state as shown in Figure 33 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.

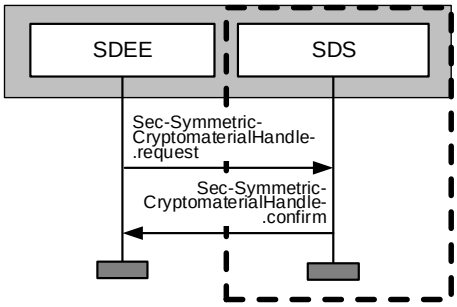


Figure 33—Process flow for obtaining a new Symmetric Cryptomaterial Handle

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
Result Code	Enumerated	Success, Failure	The result of the operation
Cryptomaterial Handle	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-GenerateKeyPair

9.3.2.1 Sec-CryptomaterialHandle-GenerateKeyPair.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-GenerateKeyPair.request (
Cryptomaterial Handle,
Algorithm
)

Name	Type	Valid range	Description
Cryptomaterial Handle	Integer	Any	A CMH in *Initialized* state
Algorithm	Enumerated type	ecdsaBrainpoolP256r1WithSha256, ecdsaBrainpoolP384r1WithSha384, 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
Result Code	Enumerated	Success, Failure	The result of the request
Public Key	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
Cryptomaterial Handle	Integer	Any	A CMH as specified in 9.2.2 in *Initialized* state
Algorithm	Enumerated type	ecdsaBrainpoolP256r1WithSha256, ecdsaBrainpoolP384r1WithSha384, ecdsaNistP256WithSha256, eciesNistp256, eciesBrainpoolP256r1	The algorithm identifier for the key pair to be stored
Public Key	Public key	Any public key valid for *Algorithm*	The public key to be stored
Private Key	Private key	Any private key valid for *Algorithm*	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
Result Code	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
Cryptomaterial Handle	Integer	Any	A CMH as specified in 9.2.2 in *Key Pair Only* state.
Certificate	1609.2 certificate	Any 1609.2 certificate containing a public verification key for the algorithm associated with *Cryptomaterial Handle*	The certificate to be stored.
Private Key Transformation	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

9.3.4.3 Function

The primitive returns the result of the corresponding request primitive.

9.3.4.3.1 Semantics of the service primitive

The parameters of the primitive are as follows:

Sec-CryptomaterialHandle-StoreCertificate.confirm (
Result Code
)

Name	Type	Valid range	Description
Result Code	Enumerated	Success Keys do not match Any *Result Code* value returned by SSME-VerifyCertificate.request	The result of the operation

9.3.4.3.2 When generated

The primitive is generated in response to Sec-CryptomaterialHandle-StoreCertificate.request.

9.3.4.3.3 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
Cryptomaterial Handle	Integer	Any	A CMH as specified in 9.2.2 in *Initialized* state
Certificate	1609.2 certificate	Any 1609.2 certificate containing a public verification key for the algorithm associated with *Cryptomaterial Handle*	The certificate to be stored
Private Key	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
Result Code	Enumerated	Success Keys do not match Any *Result Code* 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
Cryptomaterial Handle	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
Algorithm	Enumerated	aes128Ccm	An identifier of the algorithm for the symmetric key as specified in 6.3.19.
Generate	Boolean	True, False	Indicates whether the SDS is being asked to generate the key (True) or the key material is being passed (False).
Key Bytes	Octet string	An octet string of the length appropriate for the algorithm indicated in *Algorithm*; for AES-CCM, 16 bytes	The keying material to be used. Provided if *Generate* 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
Result Code	Enumerated	Success Bad parameters Failure	The result of the operation
Symmetric Cryptomaterial Handle	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
Symmetric Cryptomaterial Handle	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 *Symmetric Cryptomaterial Handle* 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
Symmetric Cryptomaterial Handle	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

9.3.8.5 Function

The primitive confirms the operation of the Sec-SymmetricCryptomaterialHandle-Delete.request primitive.

9.3.8.5.1 Semantics of the service primitive

The primitive does not take parameters.

9.3.8.5.2 When generated

The primitive is generated in response to Sec-SymmetricCryptomaterialHandle-Delete.request.

9.3.8.5.3 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),
Sign With Fast Verification,
EC Point Format,
Use Peer-to-Peer Cert Distribution,
SDEE ID (optional)
)

Name	Type	Valid range	Description
Cryptomaterial Handle	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.
Data	Octet string	Any	Used to fill in the `data` field of the SignedData-Payload.
Data Type	Enumerated	Ieee1609Dot2Data Raw	Present if and only if *Data* is present. Set to Raw if the contents of Data are to be encapsulated in an Ieee-1609Dot2Data. Set to Ieee1609Dot2Data if the contents of Data are an already-encoded Ieee1609-Dot2Data.
External Data Hash	Octet string	Length 32	Used to fill in the `extDataHash` field of the Signed-DataPayload of ToBeSignedData payload.
External Data Hash Algorithm	Enumerated	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.
PSID	Integer	$0 \ldots (2^{32} - 1)$	Used to fill in the `psid` field of the ToBeSignedData.
Set Generation Time	Boolean	True False	If True, the resulting `ToBeSignedData` contains the `generationTime` field.
Set Generation Location	Boolean	True False	If True, the resulting `ToBeSignedData` contains the `generationLocation` field.
Expiry Time	Time	Any time. If *Generation Time* is included, is later than or equal to *GenerationTime.*	If provided, the resulting `ToBeSignedData` contains the `expiryTime` field.
Signer Identifier Type	Enumerated	Certificate, digest, self	Sets the type of the `SignerIdentifier` within the `SignedData`.
Signer Identifier Certificate Chain Length	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. Ignored if *Signer Identifier Type* is not "certificate".
Sign With Fast Verification	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 `rSig` is set to `x-only`.
EC Point Format	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, EccP384CurvePoint.
Use Peer-to-Peer Cert Distribution	Boolean	True False	Whether or not to use peer-to-peer certificate distribution as specified in Clause 8. Specifically, whether or invoke SSME-Sec-OutgoingP2pcd-Info.request within this primitive
SDEE ID	Integer		Provided if *Use Peer-to-Peer Cert Distribution* 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
Result Code	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 cerificate 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
Signed Data	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-Signed-Data.request.

iv) All other the fields set as indicated by Sec-SignedData.request.

2) If the signing operation resulting from 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 Full 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
Data	Octet string	An octet string Ieee1609Dot2Data	The data to be encrypted
Data Type	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.
Data Encryption Key Type	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.
Symmetric CMH	Array of Symmetric Cryptomaterial Handles		A Symmetric CMH as specified in 9.2.3. If *Data Encryption Key Type* is "static", this parameter is present and has a single entry. Otherwise, this may or may not be present.
Recipient Certificates	Array of certificates	Any array of valid certificates which contain encryption keys	One certificate for each recipient. Not present if *Data Encryption Key Type* is "static". Optionally present if *Data Encryption Key Type* is "ephemeral".
Signed Data Recipient Info	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 6.3.33. Not present if *Data Encryption Key Type* is "static". Optionally present if *Data Encryption Key Type* is "ephemeral".
Response Encryption Key	Public key	HashedId8 of the response encryption key	Public key is for an encryption algorithm. Not present if *Data Encryption Key Type* is "static". Optionally present if *Data Encryption Key Type* is "ephemeral".
EC Point Format	Enumerated	Compressed Uncompressed	The format of the elliptic curve points included in the `RecipientInfo`s

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
Result Code	Enumerated	Success Incorrect inputs Fail on some certificates Fail on all certificates	The result of the encryption operation as specified in 5.3.4
Encrypted Data	Octet string	An Ieee1609Dot2Data of type `encryptedData`	The encrypted data, if it was created
Failed Certificates	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
Data	Octet string	An Ieee1609-Dot2Data	The data to be processed
SDEE ID	SDEE ID		The SDEE ID of the invoking SDEE
PSID	PSID		The PSID derived from context (see 5.2.3.3.2)
Use P2PCD	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 Ieee1609-Dot2Data. 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),
Geographic Region (optional),
Assurance Level (optional),
Earliest Next CRL Time
)

Name	Type	Valid range	Description	When included
Result Code	Enumerated	Success Invalid input Unknown certificate Inconsistent PSID	The result of the data extraction operation.	
Content Type	Enumerated	Unsecured Encrypted Signed	The type of the Ieee1609-Dot2Data passed in the request.	Included if *Result Code* is "success".
Service Specific Permissions	A SSP of a type specified in 6.4.29	A valid SSP according to its type (Octet string or BitmapSsp)	The SSP from the certificate that validates the signed data.	Included if *Result Code* is "success" and the certificate included a SSP with the indicated PSID.
Geographic Region	Geographic Region	An indicator of a geographic region, or "any"	An indicator of a geographic validity region	Included if *Result Code* is "success".
Assurance Level	Subject Assurance as specified in 6.4.27		The assurance level from the certificate that validates the signed data.	Included if *Result Code* is "success" and the certificate included an assurance level.
Earliest Next CRL Time	Time	Any valid time	The earliest nextCrl time value for any certificate in the chain for a signed SPDU.	Included if *Data* was of type signed and *Result Code* 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) *Geographic Region* is set only if Result Code is success and Content Type is signed. It indicates the geographic validity region of the certificate that signed the input *Data*.

e) *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.

f) *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,
Signed Data,
External Data Hash (optional),
External Data Hash Algorithm (optional),
Maximum Full Certificate Chain Length (optional),
Public Key For Self-Signed SPDU (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).
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 Full Certificate Chain Length	Integer	Any integer ≥ 2	The maximum length the certificate chain may have as specified in 5.1.2.
Public Key for Self-Signed SPDU	Public verification key	Any public verification key	The public verification key to be used to verify the signature, if the SPDU is self-signed (see 5.2.3.2.2)
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
Generation Location	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 *Signed Data* does not contain `generationLocation`.
Overdue CRL Tolerance	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 *Overdue CRL Tolerance* time ago, and has not been received, the chain is rejected.
Relevance: Expired Certificate	Boolean	True False	If True, the SDS checks that none of the certificates in the chain that signed *Signed Data* 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,
Unrecognized Id (Optional)
)

Name	Type	Valid range	Description
Result Code	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-Certificate-Chain: Inconsistent validity region 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

Name	Type	Valid range	Description
Unrecognized Id	HashedId8	Any	Provided if *Result Code* is SPDU-Parsing: Certificate not found, SPDU-Certificate-Chain: Not enough information to construct chain, or SPDU-Certificate-Chain: Chain ended at untrusted root

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`, where 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 0.

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 5.1.2.4.

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 5.1.2.4.

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 5.2.3.2.3.

p) "SPDU-Certificate-Chain: Inconsistent validity region" if for some pair of certificates the validity region in the subordinate certificate is not wholly contained in the validity region in the issuing certificate.

q) "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.

r) "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 5.1.2.4.

s) "SPDU-Consistency: Expired certificate at generation time" if the SPDU generation time is after the expiry time of the signing certificate as specified in 5.1.2.4.

t) "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.

u) "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.

v) "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 5.2.3.2.3.

w) "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.

x) "SPDU-Consistency: Unauthorized PSID" if the input *PSID* does not appear in the signing certificate as specified in 5.2.3.3.2.

y) "SPDU-Internal-Consistency: Expiry time before generation time" if the SPDU expiry time is provided and is before the SPDU generation time.

z) "SPDU-Internal-Consistency: extDataHash doesn't match" if the input *Signed Data* has a SignedDataPayload 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*.

aa) "SPDU-Internal-Consistency: no extDataHash provided" if the input *Signed Data* has a SignedDataPayload containing an `extDataHash` and the input *External Data Hash* was not provided.

bb) "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.

cc) "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*.

dd) "SPDU-Local-Consistency: Chain was too long for SDEE" if the input *Maximum Full Certificate Chain Length* was provided and if the length of the signing certificate's chain is greater than *Maximum Full Certificate Chain Length* as specified in 5.2.3.3.1.

ee) "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.

ff) "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.

gg) "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.

hh) "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.

ii) "SPDU-Relevance: Replayed 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.

jj) "SPDU-Relevance: Certificate Expired" if the input *Relevance: Expired Certificate* is True and any certificate in the chain that signed *Signed Data* has expired.

kk) If the input Signed Data fails more than one of the validity conditions, Result Code takes a value indicating one of the reasons.

If Result Code is SPDU-Parsing: Certificate not found, SPDU-Certificate-Chain: Not enough information to construct chain, or SPDU-Certificate-Chain: Chain ended at untrusted root, then the field *Unrecognized Id* contains the HashedId8 that identifies the unknown certificate, i.e. the digest field from the SignerIdentifier or the IssuerId field from the last known certificate.

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
Data	Octet string	An Ieee1609Dot2-Data of type `encryptedData`	The encrypted data to be decrypted.
Cryptomaterial Handle	Integer	Any	CMH in *Key Pair Only* or *Key and Certificate* state, where the private key is for a decryption algorithm. Alternatively, a Symmetric Crypto Material Handle.
Signed Data Recipient Info	32-byte octet string	Any	In the case where the data being decrypted was encrypted with a key obtained from a SignedData within an Ieee-1609Dot2Data, 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 Recipent 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
Result Code	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.
Data	Octet string	An Ieee1609Dot2Data of any type	If *Result Code* 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
Identifier Type	Enumerated	Certificate HashedId3 HashedId8 HashedId10	Indicates the type of input data used to identify the certificate
Identifier	Octet string	Any	The encoded certificate, HashedId3, 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-Certificate-Info.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
Result Code	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.
Certificate Data	Array of octet strings	One or more validly encoded Certificates	The certificate or certificates indicated by the corresponding request primitive.
Geographic Scope	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.
Last Received CRL Time	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".
Next Expected CRL Time	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".
Trust Anchor	Boolean	True, False	Whether or not the certificate is a trust anchor.
Verified	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.[14]
   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.

[14] In this case the status of each individual certificate in the array is not indicated; the status of a particular certificate can be obtained by invoking SSME-CertificateInfo.request with that certificate as the *Identifier* parameter.

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
Certificate	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
Certificate	1609.2 certificate	Any 1609.2 certificate	The certificate to be added
Verified	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
Result Code	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
Certificate	Certificate	A valid Certificate	The certificate to be verified
Signed SPDU	Ieee1609Dot2-Data	An Ieee1609Dot2Data of type `signedData`	If *Certificate* is an ImplicitCertificate, a signed SPDU that indicates it was signed with *Certificate*

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
Result Code	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
                    )
```

Name	Type	Valid range	Description
Identifier Type	Enumerated	Certificate HashedId8 HashedId10	Indicates the type of input data used to identify the certificate
Identifier	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
Result Code	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,
CrlCraca,
CRL Series,
Expiry
)

Name	Type	Valid range	Description
Identifiers	Array of HashedId10	As stated under Type	The HashedId10 values identifying the revoked certificates
CrlCraca	HashedId8	An octet string of length 8	An identifier for the CRACA (see 5.1.3)
CRL series	Integer	$1 \ldots 2^{32} - 1$	The CRL series that includes the revocation information
Expiry	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
Result Code	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 (
CRLCraca,
CRL Series,
RevocationInfos
)

Name	Type	Valid range	Description
CRLCraca	`HashedId8`	An octet string of length 8	An identifier for the CRACA (see 5.1.3)
CRL series	Integer	$1 \ldots 2^{32} - 1$	The CRL series that includes the revocation information
RevocationInfos	Array of RevocationInfo, each containing the entries below		
iRev	Integer	$0..2^{16} - 1$	An indication of the time period when the revocation information becomes effective
iMax	Integer	$0..2^{16} - 1$	An indication of the time period when the revocation information stops being effective
jMax	Integer	$0..2^{8} - 1$	The number of certificates within each time period
Linkage Seed 1	Octet String	Octet string of length 16	The linkage seed from the first linkage authority
Linkage Authority Identifier 1	Octet String	Octet string of length 2	An indication of the linkage authority that generated linkage seed 1, an octet string of length 2
Linkage Seed 2	Octet String	Octet string of length 16	The linkage seed from the second linkage authority
Linkage Authority Identifier 2	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
Result Code	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 (
CrlCraca,
CRL Series,
RevocationInfos
)

Name	Type	Valid range	Description
CrlCraca	`HashedId8`	An octet string of length 8	An identifier for the CRACA (see 5.1.3)
CRL series	Integer	$1 \ldots 2^{32} - 1$	The CRL series that includes the revocation information
RevocationInfos	Array of RevocationInfo, each containing the entries below		
iRev	Integer	$0..2^{16} - 1$	An indication of the time period when the revocation information becomes effective
iMax	Integer	$0..2^{16} - 1$	An indication of the time period when the revocation information stops being effective
Linkage Seed 1	Octet string	Octet string of length 16	The linkage seed from the first linkage authority
Linkage Authority Identifier 1	Octet string	Octet string of length 2	An indication of the linkage authority that generated linkage seed 1, an octet string of length 2
Linkage Seed 2	Octet string	Octet string of length 16	The linkage seed from the second linkage authority
Linkage Authority Identifier 2	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
Result Code	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
CRL Type	Enumerated value	Id Only Id Expiry	The type of the revocation information
CRL Series	Integer	$1 \ldots 2^{32} - 1$	The CRL series value from the revocation information
CRACA ID	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
Issue Date	Time	Any date prior to current date	The issue date of the revocation information
Next CRL	Time	Any time after *Issue Date*	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
CRL Series	Integer	$1 \ldots 2^{32} - 1$	The CRL series value from the revocation information
CRACA ID	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
Result Code	Enumerated	Success Unrecognized identifier Expired Not issued yet Missing	Indicates the result of the associated request
Revocation Type	Enumerated value	Hash ID based Linkage ID based	The type of the entries in the CRL
Issue Date	Time	Any date prior to current date	The issue date of the CRL
Next CRL	Time	Any time after *Issue Date*	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
SDEEs	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
Certificates	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
SDEE ID	Integer or "all"	Any	The ID of the SDEE for which this configuration applies, or "all"
p2pcd_max-ResponseBackoff	Time	Any positive time	The maximum backoff time when responding to a request
p2pcd_response-ActiveTimeout	Time	Any positive time	The time after which a response-active state ends with respect to a particular trigger certificate
p2pcd_request-ActiveTimeout	Time	Any positive time	The time after which a request-active state ends with respect to a particular trigger certificate
p2pcd_observed-RequestTimeout	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
p2pcd_currentlyUsed-TriggerCertificateTime	Time	Any positive time	The time used to determine whether a trigger certificate is "currently used".
p2pcd_response-CountThreshold	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
Result Code	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 replay 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
SDEE ID	Integer	Any	The SDEE ID that identifies the SDEE
Data	Octet string	An octet string	The encoded ToBeSignedData and signing certificate that are to be checked for being a replay
Discard Time	Time	Any time in the future	The time at which the data provided as the *Data* 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
Result Code	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
Request Active for Certificate	Boolean	True, False	Whether the SSME is in a request-active state for the certificate in the signed SPDU
Request Active for P2PCD Learning Request	Boolean	True, False	Whether the SSME is in a request-active state for the `p2pcdLearningRequest` in the signed SPDU
Response Active for P2PCD Learning Request	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 and 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[15]

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	

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. :

[15] 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.

	Amd. : Corr. :
Have any exception items been required? (See A.1.3; the answer *Yes* means that the implementation does not conform to IEEE Std 1609.2)	□Yes □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	□Yes □No
S1.1.	**Secure data exchange entity (SDEE) identification**	4.2.2.1	S1:M	□Yes □No
S1.1.1.	Support only one SDEE	4.2.2.1	S1.1:C1	□Yes □No
S1.1.2.	Distinguish between SDEEs	4.2.2.1	S1.1:C1	□Yes □No
S1.2.	**Generate secured protocol data unit (SPDU)**		S1:O2	□Yes □No
S1.2.1.	Create Ieee1609Dot2Data containing unsecured data	4.2.2.2.2	S1.2:O3	□Yes □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	□Yes □No
S1.2.2.1.	Using a valid HashAlgorithm	6.3.5	S1.2.2:M	□Yes □No
S1.2.2.1.1.	Support signing with hash algorithm SHA-256	6.3.5	S1.2.2:O3a	□Yes □No
S1.2.2.1.2.	Support signing with hash algorithm SHA-384	6.3.5	S1.2.2:O3a	□Yes □No
S1.2.2.1.3.	Support signing with other hash algorithm	6.3.5	S1.2.2:O	□Yes □No
S1.2.2.2.	Containing a Signed Data payload	6.3.6	S1.2.2:M	□Yes □No
S1.2.2.2.1.	… with payload containing data	6.3.7	S1.2.2.2:O4	□Yes □No
S1.2.2.2.2.	… with payload containing extDataHash	6.3.7	S1.2.2.2: O4	□Yes □No
S1.2.2.2.3.	… with generationTime in the security headers	6.3.9, 6.3.11	S1.2.2.2: O	□Yes □No
S1.2.2.2.4.	… with expiryTime in the security headers	6.3.9, 6.3.11	S1.2.2.2: O	□Yes □No
S1.2.2.2.5.	… with generationLocation in the security headers	6.3.9, 6.3.12	S1.2.2.2: O	□Yes □No
S1.2.2.2.6.	… with p2pcdLearningRequest in the security headers	6.3.9, 6.3.26	S1.2.2.2: O	□Yes □No

Item	Security configuration (top-level)	Reference	Status	Support
S1.2.2.2.7.	… with missingCrlIdentifier in the security headers	6.3.9, 6.3.16	S1.2.2.2: O	□Yes □No
S1.2.2.2.8.	… with encryptionKey in the security headers	6.3.9, 6.3.18	S1.2.2.2: O	□Yes □No
S1.2.2.2.8.1.	… … with a PublicEncryptionKey	6.3.9, 6.3.18, 6.3.19	S1.2.2.2.8:O5	□Yes □No
S1.2.2.2.8.2.	… … with a SymmetricEncryptionKey	6.3.9, 6.3.18, 6.3.20	S1.2.2.2.8:O5	□Yes □No
S1.2.2.3.	Support a SignerIdentifier	6.3.25	S1.2.2:M	□Yes □No
S1.2.2.3.1.	… of type digest	6.3.27	S1.2.2.3:O6	□Yes □No
S1.2.2.3.2.	… of type certificate	6.4.2	S1.2.2.3:O6	□Yes □No
S1.2.2.3.2.1.	… … maximum number of certificates included in the SignerIdentifier	6.3.25	S1.2.2.3.2 1:M > 1:O	Enter number: ( )
S1.2.2.4.	Support a Signature	6.3.29	S1.2.2:M	□Yes □No
S1.2.2.4.1.	… an ecdsa256Signature	6.3.30	S1.2.2.4:O6a	□Yes □No
S1.2.2.4.1.1.	… … using NIST p256	6.3.30	S1.2.2.4.1:O7	□Yes □No
S1.2.2.4.1.2.	… … using Brainpool p256r1	6.3.30	S1.2.2.4.1:O7	□Yes □No
S1.2.2.4.1.3.	… … with a x-only *r* value	6.3.23	S1.2.2.4.1:O8	□Yes □No
S1.2.2.4.1.4.	… … with a compressed *r* value	6.3.23	S1.2.2.4.1:O8	□Yes □No
S1.2.2.4.1.5.	… … with an uncompressed *r* value	6.3.23	S1.2.2.4.1:O8	□Yes □No
S1.2.2.4.2.	… an ecdsa384Signature using Brainpool p384r1	6.3.31	S1.2.2.4:O6a	□Yes □No
S1.2.2.4.2.1.	… … with a x-only *r* value	6.3.23	S1.2.2.4.1:O8	□Yes □No
S1.2.2.4.2.2.	… … with a compressed *r* value	6.3.23	S1.2.2.4.1:O8	□Yes □No
S1.2.2.4.2.3.	… … with an uncompressed *r* value	6.3.23	S1.2.2.4.1:O8	□Yes □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	S1.2.2:M	□Yes □No
S1.2.2.5.1.	Determine that the generation location is consistent with the region in the certificate	5.2.3.2.3, 6.4.17	S1.2.2.5:M	□Yes □No
S1.2.2.5.1.1.	Support a circularRegion	6.4.17, 6.4.18	S1.2.2.5.1:O9	□Yes □No
S1.2.2.5.1.2.	Support a rectangularRegion	6.4.17, 6.4.20	S1.2.2.5.1:O9	□Yes □No
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	□Yes □No

Item	Security configuration (top-level)	Reference	Status	Support
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	□Yes □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	□Yes □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	□Yes □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	□Yes □No
S1.2.2.5.1.4.5.	List of supported IdentifiedRegions[16]	5.2.3.4, 6.4.22	S1.2.2.5.1.4:M	Provide as Additional Information
S1.2.2.5.2.	Determine that the certificate has the proper appPermissions	6.4.8, 6.4.28	S1.2.2.5: M	□Yes □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: ( )
S1.2.2.5.3.	Maximum supported length of the full chain (sending)	5.1.2.2	S1.2.2.5: 2:M >2: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	□Yes □No
S1.2.2.7.	Support signing with explicit certificates	6.4.6	S1.2.2.5:O11	□Yes □No
S1.2.2.8.	Support signing with implicit certificates	5.3.2, 6.4.5	S1.2.2.5:O11	□Yes □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	□Yes □No
S1.2.3.	Create Ieee1609Dot2Data containing EncryptedData	4.2.2.3.2, 5.3.4, 6.3.32	S1.2:O2	□Yes □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	□Yes □No
S1.2.3.2.	Maximum number of recipients supported	6.3.32	S1.2.3 8:M > 8:O	Enter number: ( )
S1.2.3.3.	Containing PreSharedKeyRecipientInfo	6.3.33, 6.3.34	S1.2.3.2:O12	□Yes □No
S1.2.3.3.1.	Containing symmRecipientInfo	6.3.33, 6.3.35	S1.2.3.2:O12	□Yes □No
S1.2.3.3.2.	Containing certRecipientInfo	6.3.33, 6.3.36	S1.2.3.2:O12	□Yes □No
S1.2.3.3.3.	Containing signedDataRecipientInfo	6.3.33, 6.3.36	S1.2.3.2:O12	□Yes □No

[16] This list might or might not include an indication of the accuracy of the internal representation of each identified region.

Item	Security configuration (top-level)	Reference	Status	Support
S1.2.3.3.4.	Containing rekRecipientInfo	6.3.33, 6.3.36	S1.2.3.2:O12	□Yes □No
S1.2.3.4.	Support public-key encryption	6.3.38	S1.2.3:O13	□Yes □No
S1.2.3.4.1.	… using ECIES-256	6.3.38	S1.2.3.4:M	□Yes □No
S1.2.3.4.1.1.	… … using NIST p256	6.3.38	S1.2.3.4.1:O14	□Yes □No
S1.2.3.4.1.2.	… … using Brainpool p256r1	6.3.38	S1.2.3.4.1:O14	□Yes □No
S1.2.3.4.1.3.	Support encrypting to an uncompressed encryption key	6.3.18	S1.2.3.4.1:O15	□Yes □No
S1.2.3.4.1.4.	Support encrypting to a compressed encryption key	6.3.18	S1.2.3.4.1:O15	□Yes □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	□Yes □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	□Yes □No
S1.2.3.4.2.	… using a different algorithm introduced at a later date	6.3.39	S1.2.3.4:O	□Yes □No
S1.2.3.5.	Support symmetric encryption	6.3.40	S1.2.3:O13	□Yes □No
S1.2.3.5.1.	… using AES-128	5.3.8, 6.3.40	S1.2.3.5:M	□Yes □No
S1.2.3.5.2.	… using a different algorithm introduced at a later date	6.3.36	S1.2.3.5:O	□Yes □No
S1.3.	**Receive secured protocol data unit (SPDU)**		S1:O2	□Yes □No
S1.3.1.	Support preprocessing SPDUs	4.2.2.3.1	S1.3.2.3.1, S3.3 S3.4:M	□Yes □No
S1.3.2.	Verify Ieee1609Dot2Data containing Signed-Data	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	□Yes □No
S1.3.2.1.	Using a valid HashAlgorithm		S1.3.2:M	□Yes □No
S1.3.2.1.1.	Verify signed data using HashAlgorithm SHA-256	6.3.5	S1.3.2.1:O17a	□Yes □No
S1.3.2.1.2.	Verify signed data using HashAlgorithm SHA-384	6.3.5	S1.3.2.1:O17a	□Yes □No
S1.3.2.1.3.	Verify signed data using another HashAlgorithm	6.3.5	S1.3.2.1:O	□Yes □No
S1.3.2.2.	Containing a Signed Data payload	6.3.6	S1.3.2:M	□Yes □No
S1.3.2.2.1.	… with payload containing data	6.3.7	S1.3.2.2:O18	□Yes □No
S1.3.2.2.2.	… with payload containing extDataHash	6.3.7	S1.3.2.2:O18	□Yes □No
S1.3.2.2.3.	… with generationTime in the security headers	6.3.9, 6.3.11	S1.3.2.2:O	□Yes □No
S1.3.2.2.4.	… with expiryTime in the security headers	6.3.9, 6.3.11	S1.3.2.2:O	□Yes □No
S1.3.2.2.5.	… with generationLocation in the security headers	6.3.9, 6.3.12	S1.3.2.2:O	□Yes □No
S1.3.2.2.6.	… with missingCertIdentifier in the security headers	6.3.9, 6.3.26	S1.3.2.2:O	□Yes □No

Item	Security configuration (top-level)	Reference	Status	Support
S1.3.2.2.7.	… with missingCrlIdentifier in the security headers	6.3.9, 6.3.16	S1.3.2.2:O	□Yes □No
S1.3.2.2.8.	… with encryptionKey in the security headers	6.3.9, 6.3.18	S1.3.2.2:O	□Yes □No
S1.3.2.2.8.1.	… … with a PublicEncryptionKey	6.3.9, 6.3.18, 6.3.19	S1.3.2.2.8:O19	□Yes □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	□Yes □No
S1.3.2.3.	Support a SignerIdentifier	6.3.25	S1.3.2:M	□Yes □No
S1.3.2.3.1.	… of type digest	6.3.27	S1.3.2.3:O20	□Yes □No
S1.3.2.3.2.	… of type certificate	6.4.2	S1.3.2.3:O20	□Yes □No
S1.3.2.3.2.1.	… … maximum number of certificates included in the SignerIdentifier	6.3.25	S1.3.2.3.2 1:M > 1:O	Enter number: ( )
S1.3.2.4.	Support a Signature	6.3.29	S1.3.2:M	□Yes □No
S1.3.2.4.1.	… a ecdsa256Signature	6.3.30	S1.3.2.4:)O20a	□Yes □No
S1.3.2.4.1.1.	… … using NIST p256	6.3.30	S1.3.2.4.1:O21	□Yes □No
S1.3.2.4.1.2.	… … using Brainpool p256r1	6.3.30	S1.3.2.4.1:O21	□Yes □No
S1.3.2.4.1.3.	… … with a x-only *r* value	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.1.4.	… … with a compressed *r* value	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.1.5.	… … with a compressed *r* value and fast verification	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.1.6.	… … with a uncompressed *r* value	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.1.7.	… … with a uncompressed *r* value and fast verification	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.2.	… an ecdsa384Signature using Brainpool p384r1	6.3.31	S1.3.2.4:O20a	□Yes □No
S1.3.2.4.2.1.	… … with a x-only *r* value	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.2.2.	… … with a compressed *r* value	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.2.3.	… … with a compressed *r* value and fast verification	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.2.4.	… … with a uncompressed *r* value	6.3.23	S1.3.2.4.1:O22	□Yes □No
S1.3.2.4.2.5.	… … with a uncompressed *r* value and fast verification	6.3.23	S1.3.2.4.1:O22	□Yes □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	□Yes □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	□Yes □No
S1.3.2.5.1.1.	… using a circularRegion	6.4.17, 6.4.18	S1.3.2.5.1:O23	□Yes □No

Item	Security configuration (top-level)	Reference	Status	Support
S1.3.2.5.1.2.	Support a rectangularRegion	6.4.17, 6.4.20	S1.3.2.5.1:O23	□Yes □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	□Yes □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: ( )
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	□Yes □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	□Yes □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	□Yes □No
S1.3.2.5.1.6.5.	List of supported IdentifiedRegions and the accuracy of each	5.2.3.4, 6.4.22	S1.2.2.5.1.4:M	Provide as Additional Information
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:M	□Yes □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	□Yes □No
S1.3.2.5.5.	Maximum supported length of the full chain (receiving)	5.1.2.2	S1.2.2.5: 2:M >2:O	Enter number: ( )
S1.3.2.6.	Support verifying SPDUs signed with explicit authorization certificates	6.4.5	S1.3.2:O25	□Yes □No
S1.3.2.7.	Support verifying SPDUs signed with implicit authorization certificates	5.3.2, 6.4.5	S1.3.2:O25	□Yes □No
S1.3.2.8.	Support explicit certificate authority (CA) certificates	6.4.2, 6.4.6	S1.3.2:M	□Yes □No
S1.3.2.9.	Support receiving implicit CA certificates	6.4.2, 6.4.5	S1.3.2:O	□Yes □No
S1.3.2.10.	SignedData verification fails in the following circumstances:	6.3.4	S1.3.2:M	□Yes □No
S1.3.2.10.1.	… SPDU-Parsing: Invalid Input	6.3.4	S1.3.2.10:M	□Yes □No
S1.3.2.10.2.	… SPDU-Parsing: Unspported critical information field	6	S1.3.2.10:M	□Yes □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	□Yes □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	□Yes □No

Item	Security configuration (top-level)	Reference	Status	Support
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	□Yes □No
S1.3.2.10.6.	… SPDU-Certificate-Chain: Not enough information to construct chain	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.7.	… SPDU-Certificate-Chain: Chain ended at untrusted root	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.8.	… SPDU-Certificate-Chain: Chain was too long for implementation	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.9.	… SPDU-Certificate-Chain: Certificate revoked	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.10.	… SPDU-Certificate-Chain: Overdue CRL	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.11.	… SPDU-Certificate-Chain: Inconsistent expiry times	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.12.	… SPDU-Certificate-Chain: Inconsistent start times	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.13.	… SPDU-Certificate-Chain: Inconsistent chain permissions	5.1.2	S1.3.2.10:M	□Yes □No
S1.3.2.10.14.	… SPDU-Crypto: Verification failure	5.3.1	S1.3.2.10:M	□Yes □No
S1.3.2.10.15.	… SPDU-Consistency: Future certificate at generation time	5.2.3	S1.3.2.10:M	□Yes □No
S1.3.2.10.16.	… SPDU-Consistency: Expired certificate at generation time	5.2.3	S1.3.2.10:M	□Yes □No
S1.3.2.10.17.	… SPDU-Consistency: Expiry date too early	5.2.3	S1.3.2.10:M	□Yes □No
S1.3.2.10.18.	… SPDU-Consistency: Expiry date too late	5.2.3	S1.3.2.10:M	□Yes □No
S1.3.2.10.19.	… SPDU-Consistency: Generation location outside validity region	5.2.3	S1.3.2.10:M	□Yes □No
S1.3.2.10.20.	… SPDU-Consistency: Unauthorized PSID	5.2.3	S1.3.2.10:M	□Yes □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	□Yes □No
S1.3.2.10.22.	… SPDU-Internal-Consistency: extDataHash doesn’t match	5.2.3	S1.3.2.10:M	□Yes □No
S1.3.2.10.23.	… SPDU-Local-Consistency: PSIDs don’t match	5.2.3	S1.3.2.10:O	□Yes □No
S1.3.2.10.24.	… SPDU-Local-Consistency: Chain was too long for SDEE	5.2.3	S1.3.2.10:M	□Yes □No
S1.3.2.10.25.	… SPDU-Relevance: SPDU Too Old	5.2.4	S1.3.2.10:O	□Yes □No
S1.3.2.10.26.	… SPDU-Relevance: Future SPDU	5.2.4	S1.3.2.10:O	□Yes □No
S1.3.2.10.27.	… SPDU-Relevance: Expired SPDU	5.2.4	S1.3.2.10:O	□Yes □No
S1.3.2.10.28.	… SPDU-Relevance: SPDU Too Distant	5.2.4	S1.3.2.10:O	□Yes □No
S1.3.2.10.29.	… SPDU-Relevance: Replayed SPDU	5.2.4	S1.3.2.10:O	□Yes □No
S1.3.3.	Decrypt Ieee1609Dot2Data containing EncryptedData	4.2.2.3.3, 5.3.5, 6.3.32	S1.3:O17	□Yes □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	□Yes □No

Item	Security configuration (top-level)	Reference	Status	Support
S1.3.3.2.	Maximum number of RecipientInfos supported in an incoming EncryptedData	6.3.32	S1.3.3: 8:M > 8:O	Enter number: ( )
S1.3.3.2.1.	Containing symmRecipientInfo	6.3.33	S1.3.3.2:O26	□Yes □No
S1.3.3.2.2.	Containing certRecipientInfo	6.3.33	S1.3.3.2:O26	□Yes □No
S1.3.3.2.3.	Containing signedDataRecipientInfo	6.3.33	S1.3.3.2:O26	□Yes □No
S1.3.3.2.4.	Containing rekRecipientInfo	6.3.33	S1.3.3.2:O26	□Yes □No
S1.3.3.2.5.	Containing pskRecipientInfo	6.3.33, 6.3.36	S1.3.3.2:O26	□Yes □No
S1.3.3.3.	Support decrypting using a public-key algorithm	6.3.38	S1.3.3:O27	□Yes □No
S1.3.3.3.1.	… using ECIES-256	6.3.38	S1.3.3.3:M	□Yes □No
S1.3.3.3.1.1.	… … using NIST p256	6.3.38	S1.3.3.3:O28	□Yes □No
S1.3.3.3.1.2.	… … using Brainpool p256r1	6.3.38	S1.3.3.3:O28	□Yes □No
S1.3.3.3.2.	… using a different algorithm introduced at a later date	6.3.39	S1.3.3.3:O	□Yes □No
S1.3.3.4.	Support decrypting using a symmetric algorithm	6.3.40	S1.3.3:O27	□Yes □No
S1.3.3.4.1.	... using AES-128	6.3.40	S1.3.3.4:M	□Yes □No
S1.3.3.4.2.	… using a different algorithm introduced at a later date	6.3.36	S1.3.3.4:O	□Yes □No

A.2.3.2 Certificate revocation list (CRL) verification entity

Item	Security configuration (top-level)	Reference	Status	Suppport
S2.	Support CRL Validation Entity	7	O1	□Yes □No
S2.1.	Correctly verify received CRL	7.4	S2:M	□Yes □No
S2.1.1.	…using hash ID-based revocation	5.1.3.5	S2.1:O29	□Yes □No
S2.1.1.1.	… of type fullHashCrl	7.3.2	S2.1.1:M	□Yes □No
S2.1.1.2.	… of type deltaHashCrl	7.3.2	O	□Yes □No
S2.1.2.	… using linkage-based revocation	5.1.3.4	S2.1:O29	□Yes □No
S2.1.2.1.	… of type fullLinkedCrl	7.3.2	S2.1.2:M	□Yes □No
S2.1.2.2.	… of type deltaLinkedCrl	7.3.2	O	□Yes □No
S2.1.2.3.	… containing individual linkage values	7.3.6	S2.1.2:M	□Yes □No
S2.1.2.4.	… containing group linkage values	7.3.6	O	□Yes □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	□Yes □No
S3.1.	Number of supported SDEEs	8.2.6	S3.3: 1:O > 1:O	Enter number: ( )
S3.2.	Support out-of-band P2PCD operations	8	S3:O30	□Yes □No
S3.3.	**Support SSME and SDS operations for out-of-band P2PCD in the requester role**	8.2.4.1.1	S3.2:O	□Yes □No
S3.3.1.	Under at least one condition, trigger request processing on receiving a trigger SPDU	8.2.4.1.1	S3.3:M	Enter description of at least one condition under which request processing is triggered ( )
S3.3.2.	Do not trigger request processing on receiving a trigger SPDU for which a request is already active	8.2.4.1.1	S3.3:M	□Yes □No
S3.3.3.	Number of simultaneously active P2PCD learning requests	8.2.4.1.1, 8.2.6	S3.3: 1:O > 1:O	Enter number: ( )
S3.3.4.	When request processing is triggered, include a P2PCD learning request in the next SPDU for the trigger SDEE except in the following exception cases	8.2.4.1.1	S3.3: M	□Yes □No
S3.3.4.1.	Do not include a P2PCD learning request if a learning request for the same certificate has been received within **`p2pcd_observedRequestTimeout`**	8.2.4.1.1	S3.3.4:O	□Yes □No
S3.3.4.2.	Only include one P2PCD learning request no matter how many learning requests have been triggered	8.2.4.1.1	S3.3.4: M	□Yes □No
S3.3.5.	Receive notifications from a P2PCDE that a P2PCD learning response has been received and use those to update the list of known certificates.	8.2.4.1.1	S3.3: M	□Yes □No
S3.4.	**Support SSME and SDS operations for out-of-band P2PCD in the responder role**	8.2.4.2.2	S3:O30	□Yes □No
S3.4.1.	Trigger response processing on receiving a P2PCD learning request	8.2.4.2.2	S3.4:M	□Yes □No
S3.4.2.	Number of simultaneously active P2PCD learning responses	8.2.4.2.2, 8.2.6	S3.4: 1:O > 1:O	Enter number: ( )
S3.4.3.	Do not trigger response processing if less than **`p2pcd_responseActiveTimeout`** has passed since last triggered	8.2.4.2.2	S3.4: M	□Yes □No
S3.4.4.	Trigger sending response after random backoff time unless threshold number of responses have been observed	8.2.4.2.2	S3.4: M	□Yes □No
S3.4.5.	Increment number of responses observed based on input from P2PCDE	8.2.4.2.2	S3.4: M	□Yes □No
S3.5.	**Support P2PCDE operations for P2PCD**	8.2.4.2.2	S3:O30	□Yes □No
S3.5.1.	Receive responses and provide to SSME	8.2.4.1.1, 8.2.4.2.2, 8.3.1	S3.5: M	□Yes □No
S3.5.2.	Send responses when triggered by SSME	8.2.4.2.2, 8.3.1	S3.5: O	□Yes □No
S3.5.3.	Send responses over WSMP	8.2.4.2.2	S3.5.2: M	□Yes □No

S3.6.	**Support inline P2PCD operations**	8	S3:O30	□Yes □No
S3.6.1.	Support inline P2PCD requester operations	8.2.4.1.2	S3.6:O	□Yes □No
S3.6.2.	Support inline P2PCD responder operations	8.2.4.2.3	S3.6:M	□Yes □No

Annex B

(normative)

ASN.1 modules

B.1 General

This Annex presents the ASN.1 structures from the body of the document, formatted as a series of ASN.1 modules. These modules have been compiled with commercial compilers and have compiled without warnings.

In the event of a conflict between the ASN.1 in this annex and the ASN.1 in the main body of this document, the ASN.1 in the main body of this document takes precedence.

B.2 1609.2 security services

B.2.1 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) major-version-2(2)}

-- Minor version: 1

--
********************************************************************************
--
-- IEEE P1609.2 Data Types
--
--
********************************************************************************

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
  CrlSeries,
  EccP256CurvePoint,
  EncryptedDataEncryptionKey,
  EncryptionKey,
  GeographicRegion,
  GroupLinkageValue,
  HashAlgorithm,
  HashedId3,
  HashedId8,
  HashedId32,
  Hostname,
  IValue,
  LinkageValue,
  Opaque,
  Psid,
  PsidSsp,
  PsidSspRange,
  PublicEncryptionKey,
  PublicVerificationKey,
  SequenceOfHashedId3,
```

```
  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) major-version-2(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,
    ...,
    inlineP2pcdRequest   SequenceOfHashedId3 OPTIONAL,
    requestedCertificate   Certificate 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,
    eciesBrainpoolP256r1   EciesP256EncryptedKey,
    ...
}

SymmetricCiphertext ::= CHOICE  {
    aes128ccm           Aes128CcmCiphertext,
    ...
}

Aes128CcmCiphertext ::= 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,
    ...,
    sha384AndDigest         HashedId8
}

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), enroll (1) } (SIZE (8)) (ALL EXCEPT {})

PsidGroupPermissions ::= SEQUENCE  {
```

```
    subjectPermissions SubjectPermissions,
    minChainLength     INTEGER DEFAULT 1,
    chainLengthRange   INTEGER DEFAULT 0,
    eeType             EndEntityType DEFAULT {app}
}

SequenceOfPsidGroupPermissions ::= SEQUENCE OF PsidGroupPermissions
SubjectPermissions ::= CHOICE  {
    explicit        SequenceOfPsidSspRange,
    all             NULL,
    ...
}

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

END
```

B.2.2 1609.2 base types

```
IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) base(1) base-types(2) major-version-2(2)}

-- Minor version: 1

--
--*****************************************************************
--  IEEE P1609.2 Base Data Types
--
--*****************************************************************
DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;


-- ---------------------------------------------------------------
--
-- Integers
--
-- ---------------------------------------------------------------


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

SequenceOfUint8  ::= SEQUENCE OF Uint8
SequenceOfUint16 ::= SEQUENCE OF Uint16

-- ---------------------------------------------------------------
--
-- OCTET STRING types
--
-- ---------------------------------------------------------------
```

```
Opaque ::= OCTET STRING


HashedId32 ::= OCTET STRING (SIZE(32))
HashedId10 ::= OCTET STRING (SIZE(10))
HashedId8  ::= OCTET STRING (SIZE(8))
HashedId4  ::= OCTET STRING (SIZE(4))
HashedId3  ::= OCTET STRING (SIZE(3))
SequenceOfHashedId3 ::= SEQUENCE OF HashedId3


-- ---------------------------------------------------------------------
--
-- 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,
    ...,
    ecdsaBrainpoolP384r1Signature EcdsaP384Signature,
}

EcdsaP256Signature ::= SEQUENCE  {
    rSig     EccP256CurvePoint,
    sSig     OCTET STRING (SIZE (32))
}

EcdsaP384Signature ::= SEQUENCE  {
    rSig     EccP384CurvePoint,
    sSig     OCTET STRING (SIZE (48))
}

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))
    }
}

  EccP384CurvePoint::= CHOICE  {
      x-only          OCTET STRING (SIZE (48)),
      fill            NULL, -- consistency w 1363 / X9.62
      compressed-y-0  OCTET STRING (SIZE (48)),
      compressed-y-1  OCTET STRING (SIZE (48)),
      uncompressed    SEQUENCE  {
          x OCTET STRING (SIZE (48)),
          y OCTET STRING (SIZE (48))
      }
  }


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

HashAlgorithm ::= ENUMERATED  {
    sha256,
    ...,
    sha384
}

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,
    ...,
    ecdsaBrainpoolP384r1      EccP384CurvePoint
}

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)),
    ...,
    bitmapSsp           BitmapSsp,
}

BitmapSsp ::= OCTET STRING (SIZE(0..31))

SspValue ::= OCTET STRING (SIZE(0..31))

SspBitmask ::= OCTET STRING (SIZE(0..31))

PsidSspRange ::= SEQUENCE  {
```

```
    psid                Psid,
    sspRange            SspRange OPTIONAL
}

SequenceOfPsidSspRange ::= SEQUENCE OF PsidSspRange

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

BitmapSspRange ::= SEQUENCE  {
    sspValue            OCTET STRING (SIZE(1..32)),
    sspBitmask          OCTET STRING (SIZE(1..32)),
}

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.3 Certificate revocation list (CRL)

B.3.1 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) major-version-2(2)}

-- Minor version: 1
```

```
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) major-version-2(2)}
;


--
--
--  CRL contents
--
--

CrlContents ::= SEQUENCE  {
    version            Uint8 (1),
    crlSeries          CrlSeries,
    crlCraca            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          SequenceOfLAGroup
}


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.3.2 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) major-version-2(2) }

-- Minor version: 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) major-version-2(2)}

  Opaque,
  Psid
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) major-version-2(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) major-version-2(2)}

;

CrlPsid ::= Psid(256)

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.3.3 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) major-version-2(2)}


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.4 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) major-version-2(2)}

-- Minor version: 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 OER.
--
--*****************************************************************

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) major-version-2(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) major-version-2(2)}
;

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 Ieee-1609Dot2Data 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
Security Profile Version	Text string	"IEEE Std 1609.2a-2017"	Indicates the version of the security profile. Shall be "IEEE Std 1609.2a-2017" for this version of the security profile
Name	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.
PSIDs	List of PSIDs	Any list of one or more PSIDs	The PSIDs to be used by SDEEs that use this profile.
Other considerations	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
Sign Data	enumerated	True	The entity signs all outgoing data (with Sec-Signed-Data.request), outputting an Ieee1609Dot2Data *d* with *d.content* indicating `signedData`.
		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.
Signed Data in Payload	Boolean	True False	If true, in the output signed SPDU which is an Ieee1609Dot2Data *d*, the field *d.content.signedData. tbsData.payload.data* is present. If false, it is absent.
External Data	Boolean	True False	If true, in the output signed SPDU which is an Ieee1609Dot2Data *d*, the field *d.content.signedData. tbsData.payload.extDataHash* is present. If false, it is absent.
External Data Source	Text		How the SDEE obtains any data that is hashed to provide `extDataHash`.
External Data Hash Algorithm	Enumerated	"SHA-256"	The algorithm used to hash the external data.
Set Generation Time in Security Headers	Boolean	True False	The value set as *Set Generation Time* when invoking Sec-SignedData.request. If True, in the output signed SPDU which is an Ieee1609Dot2Data *d*, the field *d.content.signedData.tbsData.headerInfo. .generationTime* is present. If false, it is absent.
Set Generation Location in Security Headers	Boolean	True False	The value set as *Set Generation Location* when invoking Sec-SignedData.request. If True, in the output signed SPDU which is an Ieee1609Dot2Data *d*, the field *d.content.signedData.tbsData.headerInfo .generationLocation* is present. If false, it is absent.
Set Expiry Time in Security Headers	Boolean	True False	The value set as *Set Expiry Time* when invoking Sec-SignedData.request. If True, in the output signed SPDU which is an Ieee1609Dot2Data *d*, the field *d.content.signedData.tbsData.headerInfo- .expiryTime* is present. If false, it is absent.
Signed SPDU Lifetime	Time interval	Any time interval, or "n/a" if *SetExpiryTimeIn-SecurityHeaders* 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 *Set Expiry Time in Security Headers* is True. In this case, the field *d.content.signedData. tbsData.headerInfo.expiryTime* takes the value (current time + *Signed SPDU Lifetime*).
Signer Type Self	Enumerated	"Required", "Permitted", "Prohibited"	Whether in the Ieee1609Dot2Data *d*, the field *d.- content.signedData.signer* may take the value self.
Signer Type Self Permitted	Text		If *Signer Type Self* is equal to Permitted, the conditions under which the signer type may be self.
Verification Key Location for Signer Type Self	Text		If *Signer Type Self* is equal to Permitted or Prohibited, instructions for how the SDEE can obtain the verification key.
Signer Identifier Policy Type	Enumerated	Simple Text	Describes the type of the Signer Identifier Policy. In the output signed SPDU, which is an Ieee1609Dot2-Data *d*, the Signer Identifier Policy indicates which option in the field *d.content.signedData.signer* 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
Simple Signer Identifier Policy: Minimum Inter Cert Time	Time interval (for example, "one second")	Any valid interval of time, or "always"	Used to set *Signer Identifier Type* when invoking Sec-SignedData.request, i.e., indicates which option in the field *d.content.signedData.signer* is selected. 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 *certificate* 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 *Signer Identifier Type* set to "*certificate*" and *Signer Identifier Cert Chain Length* set to *Simple Signer Identifier Policy: Cert ChainLength*. In terms of the output, the field *d.-content.signedData.signer*. *certificate* is present and contains (*Simple Signer Identifier Policy: Cert ChainLength*) certificates. Otherwise, the Sec-SignedData.request primitive is invoked with *Signer Identifier Type* set to *digest* and in the output Ieee1609Dot2Data *d*, the field *d.content.signedData.signer.digest* is present.
Simple Signer Identifier Policy: Exceptions	Boolean	True False	If True, there are exceptions to the simple policy which are recorded in the notes. If False, there are no exceptions.
Simple Signer Identifier Policy: Signer Identifier Cert Chain Length	Integer or enumerated	−256 to −1 1 to 256 "Max"	The value set as the *Signer Identifier Cert Chain Length* when invoking Sec-SignedData.request; in other words, the intended length of the certificate chain to be sent.
Text Signer Identifier Policy	Text	Human-readable text	A text description of how the *Signer Identifier Type* is set, i.e., which option in the field *d.content. signedData.signer* is selected.
Sign With Fast Verification	enumerated	Uncompressed Compressed No Optional	The value set as *Sign With Fast Verification* 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. In terms of the output Ieee1609Dot2Data *d*: if this value is "Uncompressed", the field *d.content. signedData.signer.signature.[ecdsa256signature*| *ecdsaBrainpoolP256r1Signature*| *ecdsaBrainpoolP384r1Signature].r* indicates *uncompressed*; if this value is "compressed", that field indicates *compressed-y-0* or *compressed-y-1*; if it is "no", that field indicates *x-only*; if it is "optional", the field may indicate any of the choices.
EC Point Format	Enumerated	Uncompressed Compressed Variable	The value set as the *EC Point Format* when invoking Sec-SignedData.request. In terms of the output Ieee1609Dot2Data *d*: if this is "Uncompressed", then any elliptic curve point fields in *d* indicate the choice *uncompressed*; if this is "Compressed", then any elliptic curve point fields in *d* indicate the choice *compressed-y-0* or *compressed-y-1*.
p2pcd_flavor	Enumerated	Inline Out of Band None	Whether to use the peer-to-peer certificate distribution defined in Clause 8.

Name	Type	Recommended values	Description
p2pcd_max-ResponseBackoff	Time or n/a		If *p2pcduseInteractiveForm* is True, the maximum backoff time when responding to a request as defined in Clause 8. Otherwise, "n/a".
p2pcd_response-ActiveTimeout	Time or n/a		If *p2pcduseInteractiveForm* 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".
p2pcd_request-ActiveTimeout	Time or n/a		If *p2pcduseInteractiveForm* 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".
p2pcd_observed-RequestTimeout	Time or n/a		If *p2pcduseInteractiveForm* 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".
p2pcd_currentlyUsed-TriggerCertificate-Time	Time or n/a		If *p2pcduseInteractiveForm* is True, a time used to determine whether a trigger certificate is "currently used" as defined in Clause 8. Otherwise, "n/a".
p2pcd_response-CountThreshold	Integer or n/a		If *p2pcduseInteractiveForm* 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".
Repeat Signed SPDUs	Boolean	True False	Whether each new PDU is signed or a signed SPDU is repeated for its lifetime.
Time Between Signing	Time or n/a		If *Repeat Signed SPDUs* is True, the time between generating fresh signed SPDUs.
Encrypt Data	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 Certifiate 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 SDEE can predict, but a receiving SDEE 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 Type Self*, *Signer Type Self Permitted*, *Verification Key Location for Signer Type Self*: In general, *Signer Type Self* should be "Prohibited" and the other two fields can be omitted.

— *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. Note that the *Simple Signer Identifier Policy: Signer Identifier Cert Chain Length* is the number of certificates that will be sent along with a signed PDU; it is not the maximum certificate chain length of the end entity itself. The receiving side has a policy establishing what the maximum number is for this value. For settings where predistribution of CA certificates is not possible and 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_flavor* and the interactive-form *p2pcd_** variables: in general it is recommended that SDEEs use P2PCD if practical. 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 WAVE Security Services instance will send a request without regard to whether or not other instances 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.

NOTE—The Simple Signer Identifier Policy: Signer Identifier Cert Chain Length is the number of certificates that will be sent along with a signed PDU; it is not the maximum certificate chain length of the end entity itself. The receiving side has a policy establishing what the maximum number is for this value.

C.2.1.3.2 Receiving

This part of the IEEE 1609.2 security profile contains the following information.

Name	Type	Valid range	Description
Use Preprocessing	Enumerated	True False Text	Specifies whether or not a receiving SDEE invokes Sec-SecureDataPreprocessing.confirm. This should be set to "False" if *Sign Data* 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 `digest`. It should also be set to "True" if *p2pcd_flavor* takes any value other than "none" 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.
Verify Data	Enumerated	True False Text	Specifies whether or not a receiving entity attempts to verify data. If *SignData* in the sending profile is False, this is set to "False". If *SignData* 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.
Maximum Full Certificate Chain Length	Integer	Integer ≥ 2	The value set as *Maximum Full Certificate Chain Length* when invoking Sec-SignedData-Verification.request.
Relevance: Replay	Boolean	True False	The value set as *Relevance: Replay* when invoking Sec-SignedDataVerification.request.
Relevance: Generation Time in Past	Boolean	True False	The value set as *Relevance: Generation Time in Past* to Sec-SignedDataVerification.request.
Validity Period	Time interval	Any time interval such as "5 seconds" or "between 1 and 2 seconds"	The value to set as *Validity Period* when invoking Sec-SignedDataVerification.request. Set if *Relevance: Generation Time in Past* or *Relevance: Replay* 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 that *Validity Period* are rejected as invalid.

Name	Type	Valid range	Description
Relevance: Generation Time in Future	Boolean	True False	The value set as *Relevance: Generation Time in Future* to Sec-SignedDataVerification.request.
Acceptable Future Data Period	Time	Any positive time value	The value set as *Acceptable Future Data Period* when invoking Sec-SignedData-Verification.request, or the algorithm for setting that value.
Generation Time Source	Enumerated	Security Headers Payload	If *GenerationTimeSource* is "Security Headers", the generation time parameters to Sec-Signed-DataVerification.request is obtained from the corresponding field in the HeaderInfo of the SPDU. If *GenerationTimeSource* is "Payload", the generation time is obtained from elsewhere (for example, from the payload) and are provided by the entity to Sec-SignedDataVerification.request.
Relevance: Expiry Time	Boolean	True False	The value set as *Expiry Time Relevance Check* when invoking Sec-SignedData-Verification.request.
Expiry Time Source	Enumerated	This need only be set if *Expiry Time Relevance Check* is true.	
		Security Headers	The *Expiry Time* parameter to Sec-SignedData-Verification.request is obtained from the corresponding fields in Sec-SecureDataPre-processing.confirm.
		Payload	The *Expiry Time* 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 *Reject Distant Messages* when invoking Sec-SignedData-Verification.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 *Validity Distance* when invoking Sec-SignedDataVerification.request. Set only if *Reject Distant Messages* is "True".
Generation Location Source	Enumerated	This is specified if *Reject Distant Messages* or *GenerationLocationConsistencyCheck* is true, or *SupportedGeographicRegions* (see C.2.1.3.3) contains any value other than "None".	
		"Security Headers"	The *Generation Location* parameter to Sec-SignedDataVerification.request is obtained from the corresponding fields in Sec-SecureDataPre-processing.confirm.
		"Payload"	The *Generation Location* is obtained from the payload of the PDU and is provided by the entity to Sec-SignedDataVerification.request.
		"Other"	The *Generation Location* 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.

Name	Type	Valid range	Description
Additional Geographic Consistency Conditions	Boolean	True False	If True, then additional geographic consistency conditions need to be checked to determine the validity of a signed SPDU as described in 5.2.3.3.5. These consistency conditions are not part of the security profile but are expected to be provided as part of the SDEE specification
Identified Region Representation Accuracy	Text or n/a	A description of the accuracy requirements for identified region used by the SDEE, if appropriate	As discussed in 5.2.3.4, this may be a list of the identified region types or individual identified regions that are used by the SDEE, along with a description of the required accuracy of the internal representation of each identified region. The description may provide different accuracy requirements for different regions. The description may also state that the accuracy requirement can be determined on a per-site or per-deployment basis.
Overdue CRL Tolerance	Time period or text	Any positive time period or text	The value set as *Overdue CRL Tolerance* 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.

— *Additional Geographic Consistency Conditions*: Should be set to "True" if it is appropriate to include additional consistency conditions governing whether or not a signed SPDU is authorized to make statements relating to a particular geographic location, as discussed in 5.2.3.3.5.

— *Identified Region Representation Accuracy*: This is a trade-off between the cost of storing accurate representations of the regions and the risk that a compromised SDEE will attempt to send from a location that it is not entitled to send from, but appears entitled to send from due to map inaccuracies. For land borders it may be wise to require representations to accurately represent which roads lie in a region, while it may not be necessary to require strict accuracy for a border that lies between roads. Accuracy requirements might additionally be different for sea borders.

— *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
Signing Key Algorithm	Enumerated	ecdsaNistP256withSha256 ecdsaBrainpoolP256r1withSha256	One of the valid signing algorithms identified in 5.3.1 and 6.4.38.
Encryption Algorithm	Enumerated	eciesNistP256 eciesBrainpoolP256r1	One of the valid encryption algorithms identified in 5.3.5 and 6.3.20.
Implicit or Explicit Certificates	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.
EC Point Format	Enumerated	Compressed Uncompressed	How points are to be represented in certificates.
SupportedGeographic Regions	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.
Maximum Full Certificate Chain - Length	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 full 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.
Use Individual Linkage ID	Boolean	True False	Whether to support individual linkage ID-based revocation in the certificates.
Use Group Linkage ID	Boolean	True False	Whether to support group linkage ID-based revocation in the certificates.
Signature Algorithms in Chain or CRL	Sequence of Enumerated	One or more of: ecdsaNistP256withSha1 ecdsaBrainpoolP256r1withSha1 ecdsaBrainpoolP384r1withSha1	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[17]

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".

[17] 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
Name		
PSIDs		
Other considerations		

C.3.2.2 Sending

Field	Value	Notes
Sign Data		
Signed Data in Payload		
External Data		
External Data Source		
External Data Hash Algorithm		
Set Generation Time in Security Headers		
Set Generation Location in Security Headers		
Set Expiry Time in Security Headers		
Signed SPDU Lifetime		
Signer Type Self		
Signer Type Self Permitted		
Verification Key Location for Signer Type Self		
Signer Identifier Policy Type		
Simple Signer Identifier Policy: Minimum Inter Cert Time		
Simple Signer Identifier Policy: Exceptions		
Simple Signer Identifier Policy: Signer Identifier Cert Chain Length		
Text Signer Identifier Policy		
Sign With Fast Verification		
EC Point Format		
p2pcd_flavor		
p2pcd_maxResponseBackoff		
p2pcd_responseActiveTimeout		
p2pcd_requestActiveTimeout		
p2pcd_observedRequestTimeout		
p2pcd_currentlyUsedTriggerCertificateTime		
p2pcd_responseCountThreshold		
Repeat Signed SPDUs		
Time Between Signing		
Encrypt Data		

C.3.2.3 Receiving

Field	Value	Notes
Use Preprocessing		
Verify Data		
Maximum Full Certificate Chain Length		
Relevance: Replay		
Relevance: Generation Time in Past		
Validity Period		
Relevance: Generation Time in Future		
Acceptable Future Data Period		
Generation Time Source		
Relevance: Expiry Time		
Expiry Time Source		
Consistency: Generation Location		
Relevance: Generation Location Distance		
Validity Distance		
Generation Location Source		
Additional Geographic Consistency Conditions		
Identified Region Representation Accuracy		
Overdue CRL Tolerance		
Relevance: Certificate Expiry		
Encrypted Data		

C.3.2.4 Security management

Field	Value	Notes
Signing Key Algorithm		
Encryption Algorithm		
Implicit or Explicit Certificates		
EC Point Format		
Supported Geographic Regions		
Maximum Full Certificate Chain Length		
Use Individual Linkage ID		
Use Group Linkage ID		
Signature Algorithms in Chain or CRL		

C.3.2.5 Other

Field	Value	Notes
Fields that may be subject to policy update		

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.

SDEE specifiers may choose to use SSPs that are opaque or in the form of bitmaps of bitmaps (in the end entity certificate) and bitmasks (in the CA certificate). No matter what form is used, the responsibility is still with the SDEE to define the semantics of the SSP, i.e. how it maps to the permissions of associated communications. The difference between the SSP forms from the point of view of the SDS lies in how consistency is checked between certificates in the chain; in particular, if there is a CA certificate that can issue certificates for some but not all of the SSP values associated with a particular PSID. The opaque approach offers most flexibility to the SSP specifier, but with this approach the only way to encode multiple SSP values in a CA certificate is by explicitly listing them. The bitmap approach allows for very compact encodings of multiple SSP values in a CA certificate but requires that it is possible for the application permissions to be sensibly expressed as a bitmap, i.e. that they are more-or-less independent yes/no choices.

SDEE specifiers may take these considerations into account when determining the SSP format for their SDEE specification.

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.

C.7 Source of encryption keys

This standard supports three means for a sending SDEE to obtain encryption public keys to produce an encrypted SPDU:

— From a certificate.

— From the encryptionKey field in the HeaderInfo of a SignedData.

— By some other means.

In the first two cases, the key identifier in the RecipientInfo is calculated by hashing the "container" (the certificate or the signed SPDU); in the third case, the key identifier is calculated by hashing only the public key. The advantage of hashing the "container" is to prevent misbinding attacks. In these attacks an attacker tricks one victim into encrypting a message that the victim thinks is meant for one party but is in fact sent to another party. For example, say Alice has a public key. Mallory sends this public key to Bob, and Bob encrypts a message, thinking it's for Mallory. Mallory then forwards the encrypted message to Alice; Alice decrypts it and thinks that it is intended for her because it was encrypted with her encryption key.

This attack is thwarted by hashing the container: in the above case, whether Mallory had managed to get Alice's public key issued as the encryption key in a certificate for Mallory, or instead had included it in a signed SPDU, the hashed container would be identified with Mallory. Alice would expect that if a message was intended for her, the hashed container would be her certificate or a signed SPDU that she had previously sent, and so the attempt to persuade her that the encrypted SPDU was encrypted for her would fail because (a) the container hash in the RecipientInfo would not match any container hash that Alice had stored; and (b) the container hash that Alice provides as parameter P1 to ECIES, as specified in 5.3.5, would not match the container hash that Bob used when encrypting.

It is therefore recommended that SDEE designers who use public key encryption make use of either public keys in certificates or public keys in signed SPDUs, and avoid "raw" public keys because they do not mitigate this misbinding threat.

For an SDEE designer choosing between using a public key from a certificate or a public key from a signed SPDU:

— If the public key is in a certificate, there is one long-term decryption key. This makes key management and storage simpler on the device, but it carries the risk that if the decryption key is compromised, all encrypted SPDUs encrypted with that key can be decrypted. In this scenario the lifetime of the decryption key is essentially the lifetime of the certificate, so if the device is physically compromised in that time, then a significant number of past communications could be revealed.

— If the public key is in a signed SPDU, there may during the course of its lifetime be many decryption keys to be managed by any device that hosts SDEEs that sign SPDUs with encryption keys and receive the encrypted SPDUs for decryption. The device will need to store each individual decryption key along with the canonicalized hash of the signed SPDU that contained the corresponding encryption key for at least the length of time in which it expects to receive responses. This creates more key management complexity than is the case for encryption keys in certificates. However, the advantage is that if one decryption key is compromised, only messages encrypted with that key will be compromised. In this scenario an encryption key may be expected to be used one time only, and the corresponding decryption key can be deleted once the encrypted SPDU has been decrypted. This

provides greater protection for past messages in the event of device compromise than is provided by the alternative model of long-lived encryption keys in certificates.

The SDEE designer may select whether encryption keys are contained in certificates or signed SPDUs taking the above considerations into account.

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 SPDU 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 SPDU. 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 SPDU if that signature verifies.

d) A receiving SDEE need not carry out all validity checks on a given SPDU 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 SPDU is valid, this does not demonstrate that the SPDU 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 SPDU.

        i) **Provider service identifier (PSID)**. A receiving SDEE should determine that the payload of the signed SPDU 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 SPDU 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 `crlCraca` 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 `crlCraca` 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 6.3.25.

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 `crlCraca` 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 1, i.e. containing two certificates. 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.

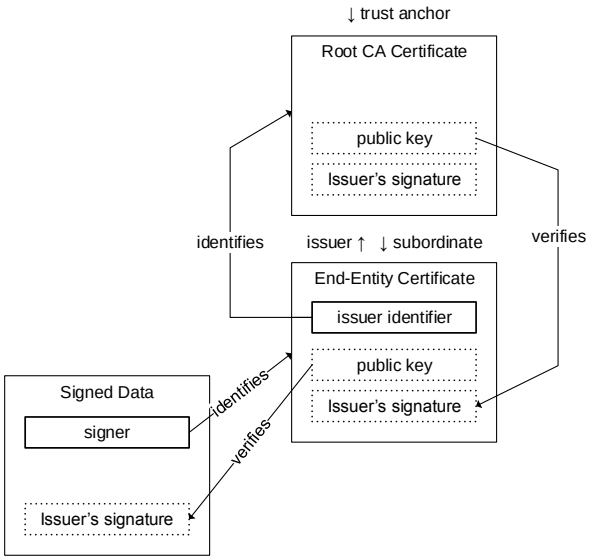


Figure D.1—Length-1 certificate chain with explicit certificates and a root CA as trust anchor

Figure D.2 shows a certificate chain of length 1, i.e. containing two certificates, 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.

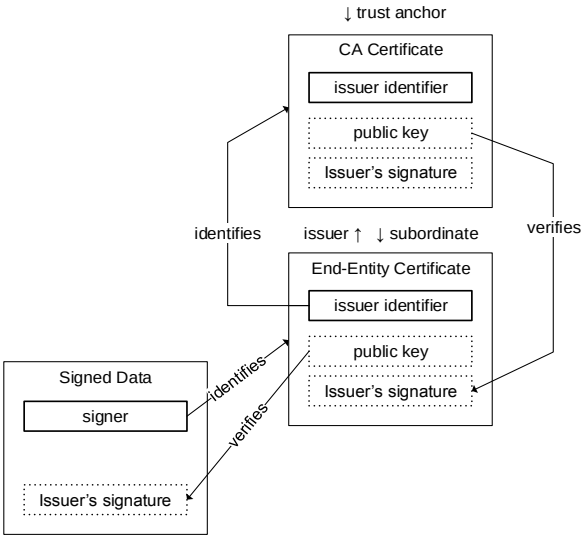


Figure D.2—Length-1 certificate chain with explicit certificates and a non-root CA as trust anchor

Figure D.3 shows a certificate chain of length 1, i.e. containing two certificates, 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.

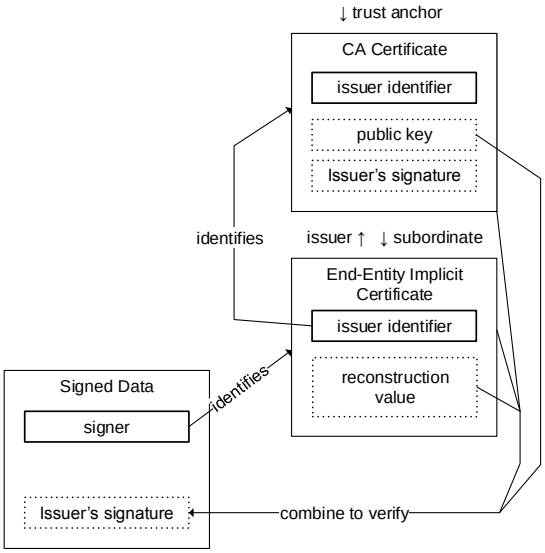


Figure D.3—Length-1 certificate chain with implicit certificates and a non-root CA as trust anchor

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.

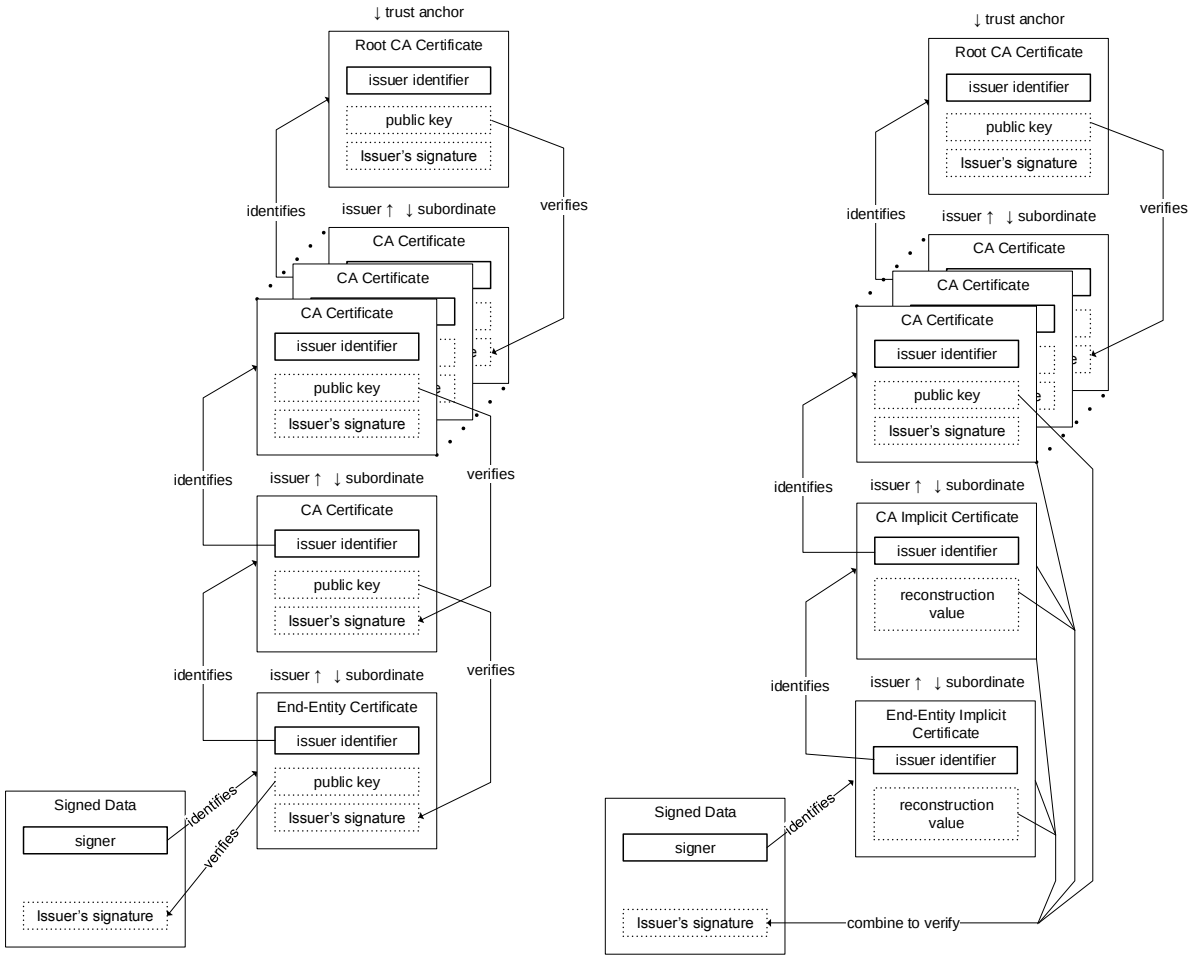


Figure D.4—Long certificate chains, one with all explicit certificates (left) and one ending with two implicit certificates (right)

D.3.2 Construction

A signed SPDU need not include all the certificates in the chain. Instead, the SPDU may include a chain of one or more certificates that does not reach all the way back to a trust anchor or the SPDU 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 certficates, 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.

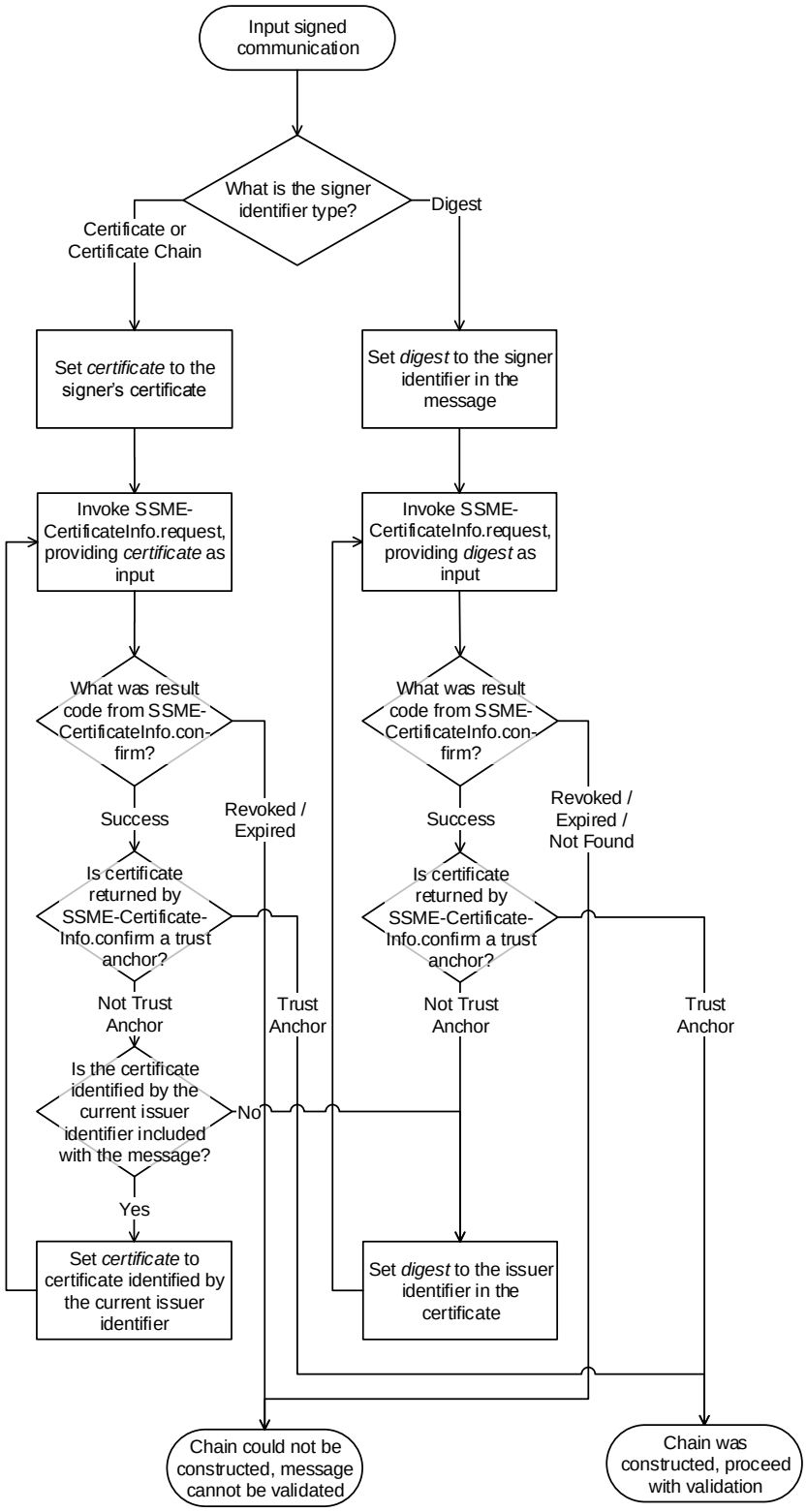


Figure D.5—Logic flow for constructing a certificate chain

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

Subclase 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 *isRequestActive*(*c*, *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 and will expire in less time than `p2pcd_observedRequestTimeout`(*SDEE ID*), the SSME initializes (or re-initializes) that the timerwith 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 *recentlyUsed-SigningCertificates*(*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.

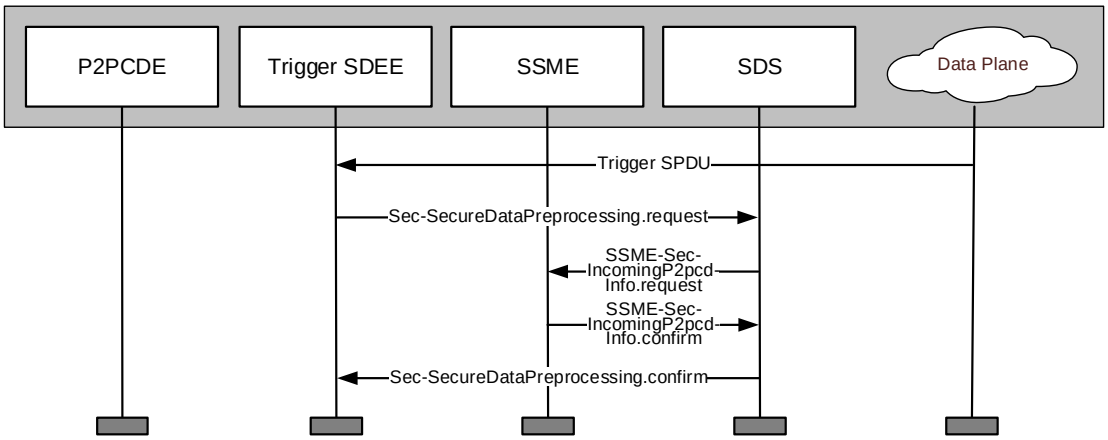


Figure D.6—P2PCD operations: receiving SPDU for trigger SDEE

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*, 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 HeaderInfofor 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.

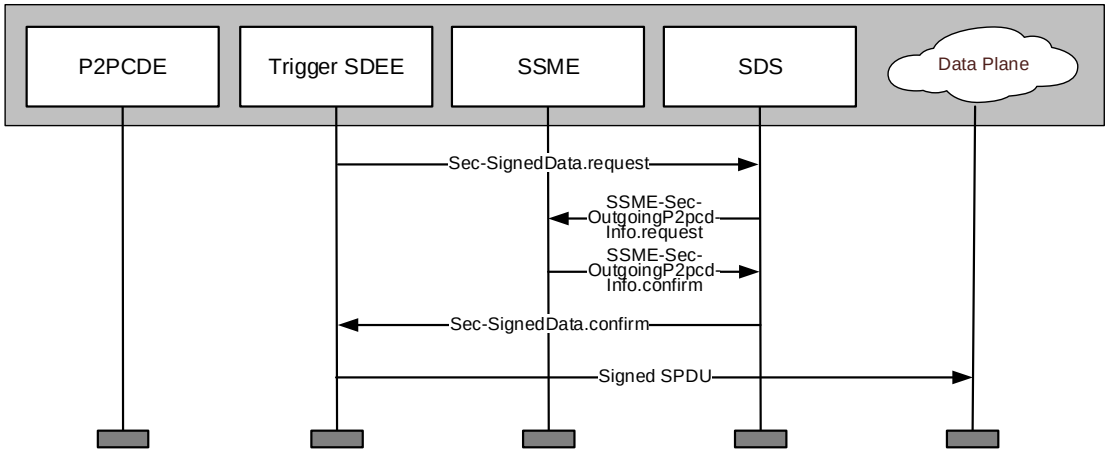


Figure D.7—P2PCD operations: sending SPDU for trigger SDEE

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.

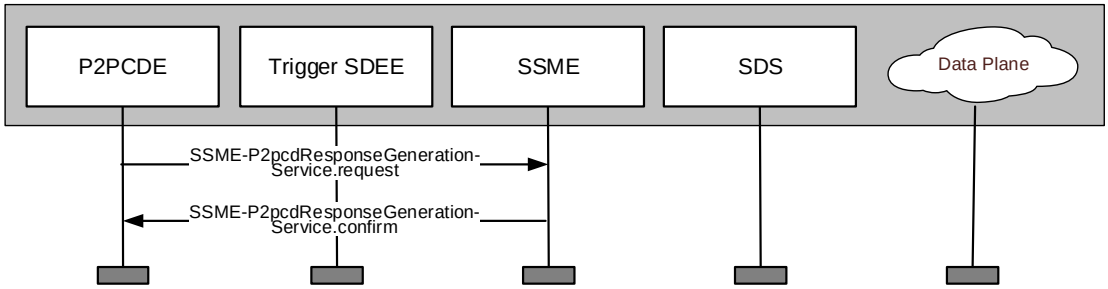


Figure D.8—P2PCD operations: P2PCD Entity registers for response generation service

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-P2pcdResponse-Generation.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.

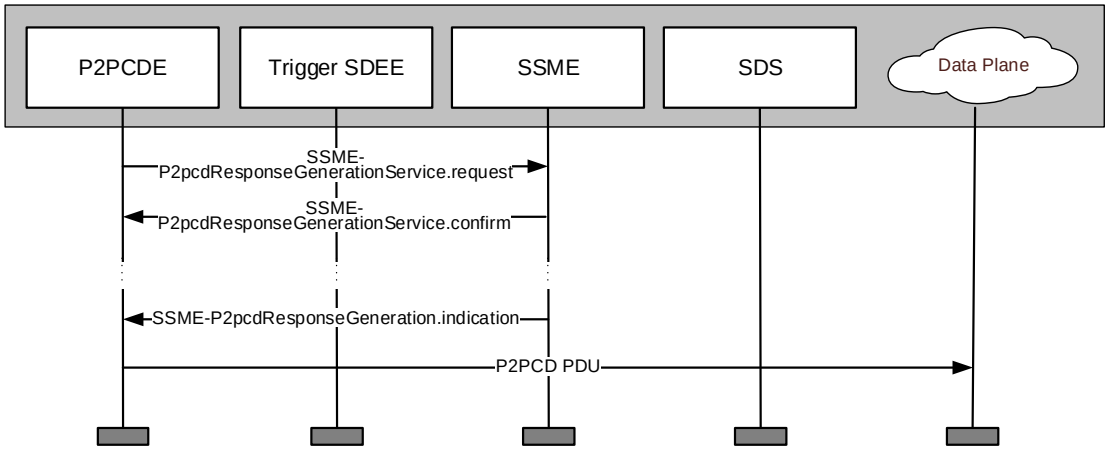


Figure D.9—P2PCD operations: P2PCD Entity sends response

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.

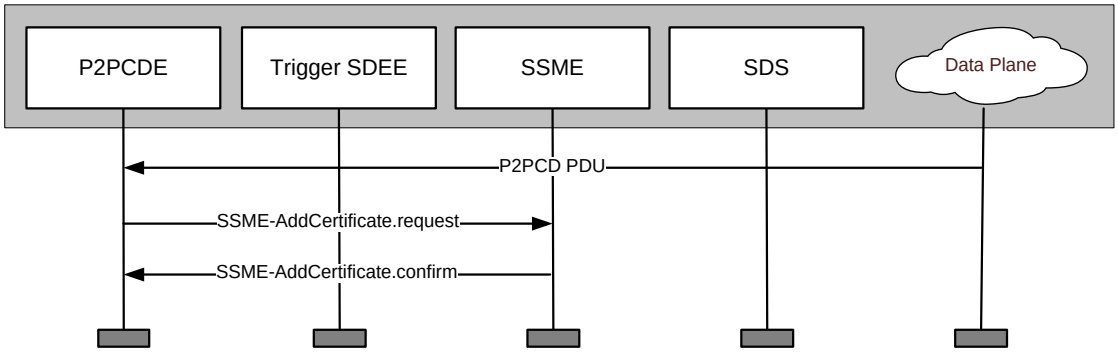


Figure D.10—P2PCD operations: P2PCD Entity receives response

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 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 *digest*. The payload is 15 bytes long and the entire SPDU 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 : '546869732069732061204253 4D0D0A'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 : '546869732069732061204253 4D0D0A'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 :
'919293949596979891929394959697989192939495969798919293949596979 8'H
        }
      }
    },
    signature ecdsaNistP256Signature : {
      r compressed-y-0 :
'313233343536373831323334353637383132333435363738'H,
      s
'414243444546474841424344454647484142434445464748414243444546474 8'H
    }
  }
}
```

D.5.3 PsidGroupPermissions examples

— An enrollment certificate contains a `certRequestPermissions` field containing an instance of this type with `minChainLength` equal to 0, `chainLengthRange` equal to 0, and `eeType` equal to `app` (because the enrollment certificate is used to request authorization certificates).

— A certificate for a CA that directly issues end-entity certificates might contain a `certRequestPermissions` field containing an instance of this type for a given PSID/SSP combination with `minChainLength` equal to 1, `chainLengthRange` equal to 0, and `eeType` equal to `app`. This indicates that it is entitled to issue end-entity certificates for that PSID/SSP combination.

— A certificate for an intermediate CA might contain a `certRequestPermissions` field containing an instance of this type for a given PSID/SSP combination with `minChainLength` equal to 2, `chainLengthRange` equal to 0, and `eeType` equal to `app`. This indicates that there must be exactly one CA in the chain between the intermediate CA and the end-entity.

— A certificate for a root CA might have an instance of this field for a given PSID/SSP combination with `minChainLength` equal to 3, `chainLengthRange` equal to −1, and `eeType` equal to (`app`, `enroll`). This indicates that there must be at least two CAs in the chain between the root CA and the end-entity (`minChainLength` = 3) and that there may be any number greater than or equal to two (`chainLengthRange` = −1, i.e. the length of the chain is not constrained so long as it is greater than or equal to `minChainLength`).

D.5.4 Root CA Certificate Profile

This section contains an example V2X root CA certificate profile for which the following hold:

— It is self-signed (issuer = `self`).

— This certificate will not be revoked (`cracaId` of all 0s AND `CrlSeries` value of 0).

— This certificate is valid worldwide because region is absent and issuer is `self`.

— Application Permissions: There are two application-level permissions (PSIDs) associated with the root certificate:

  — Security Management (issuance of certificates).

  — CRL Issuance – This root CA is also the CRACA and its certificate indicates that there is a single CRL series associated with it.

— Issuance Permissions: This root certificate's issuance rights are constrained as follows:

  — It can issue any permissions.

  — Either end entity application or enrollment certificates may chain to it.

  — `minChainLength` is 3, universally. This means that there must be two CA layers between it and end entity certificates no matter the PSID.

  — `chainLengthRange` is -1, universally. This means that the certificate chain to the end entities (from this root) may be any length equal to or greather than `minChainLength` which is 3.

  — For the Security Management, Misbehavior Reporting and CRL issuance PSIDs, it may issue any permissions to a certificate directly under it (`minChainLength` of 1).

- SspRange values that are absent also indicate “all”, meaning any certificate permissions may be issued from this root.

- Example Populated Variables:
  - Validity Period Start: 385689600
  - RootCaCertExpiration: 70 Years
  - ScmsSpclComponentCrlSeries: 256
  - SecurityMgmtPsid: 35
  - MisbehaviorReportingPsid: 38
  - CrlPsid: 256

```
RootCaCertificate ::= ExplicitCertificate (WITH COMPONENTS { ...,
  issuer (WITH COMPONENTS {self}),
   toBeSigned (WITH COMPONENTS { ...,
      id (WITH COMPONENTS {
      name ("v2xrootca.ghsiss.com")
      }),
      cracaId('000000'H),
      crlSeries(0),
      validityPeriod (WITH COMPONENTS { ...,
        duration (RootCaCertExpiration)
      }),
      region ABSENT,
      assuranceLevel ABSENT,
      appPermissions (SequenceOfPsidSsp (SIZE(2)) (CONSTRAINED BY {
        PsidSsp (WITH COMPONENTS {
          psid (SecurityMgmtPsid),
          ssp --OER encoding of ScmsSsp indicating RootCaSsp
        }),
        PsidSsp (WITH COMPONENTS {
          psid (CrlPsid),
          ssp (WITH COMPONENTS {opaque(CONTAINING CrlSsp (WITH
COMPONENTS
          {...,
            associatedCraca(isCraca),
            crls (PermissibleCrls (SIZE(1)) (CONSTRAINED BY {
              CrlSeries (ScmsSpclComponentCrlSeries
            }))
          }))})
        })
      })),
      certIssuePermissions (SequenceOfPsidGroupPermissions (SIZE(4))
       (CONSTRAINED BY {
        PsidGroupPermissions ( WITH COMPONENTS {...,
          subjectPermissions (WITH COMPONENTS {all }),
         minChainLength(3),
         chainLengthRange(-1),
         eeType ({app, enroll})
        }),
        PsidGroupPermissions ( WITH COMPONENTS {...,
          subjectPermissions (WITH COMPONENTS{
            explicit (SequenceOfPsidSspRange (SIZE (1)) (WITH COMPONENT
```

```
            (WITH COMPONENTS {
             psid (SecurityMgmtPsid),
             sspRange ABSENT
            }))))
          }),
          minChainLength(1),
          chainLengthRange(-1),
          eeType ({app, enroll})
        }),
        PsidGroupPermissions ( WITH COMPONENTS {...,
          subjectPermissions (WITH COMPONENTS{ explicit
(SequenceOfPsidSspRange
            (SIZE (1)) (WITH COMPONENT (WITH COMPONENTS {
             psid (MisbehaviorReportingPsid),
             sspRange ABSENT
             }))))
          }),
          minChainLength(1),
          chainLengthRange(-1),
          eeType ({app, enroll})
        }),
        PsidGroupPermissions ( WITH COMPONENTS {...,
          subjectPermissions (WITH COMPONENTS{ explicit
(SequenceOfPsidSspRange
            (SIZE (1)) (WITH COMPONENT (WITH COMPONENTS {
             psid (CrlPsid),
             sspRange (WITH COMPONENTS {all})
             }))))
          }),
          minChainLength(1),
          chainLengthRange(-1),
          eeType ({app, enroll})
        })
    })),
    certRequestPermissions ABSENT,
    canRequestRollover ABSENT,
    encryptionKey ABSENT,
    verifyKeyIndicator (WITH COMPONENTS {
      verificationKey (WITH COMPONENTS {
        ecdsaNistP256 (WITH COMPONENTS {
          compressed-y-0, compressed-y-1
        })
      })
    })
  })
})
```

D.6 Cryptographic Test Vectors

D.6.1 AES-CCM-128

==========

It is based on NIST SP 800-38C (and RFC 3610) with the following:

- Adata = 0, i.e. no associated authenticated data

- t=16, i.e. tag length is 16 octets

- n=12, i.e. Nonce length is 12 octets

- q=3, i.e. the message length in octets is encoded in 3 octets

Inputs:

- key: {octet string} AES-CCM key, K (hex encoded bytes)

- nonce: {octet string} nonce, N (hex encoded bytes)

- plaintext: {octet string} plaintext to be encrypted and authenticated, P (hex encoded bytes)

Output:

ciphertext || tag = C || T {octet string}

Test Vector #1:

K = 0xE58D5C8F8C9ED9785679E08ABC7C8116

key[16] =

{ 0xE5, 0x8D, 0x5C, 0x8F, 0x8C, 0x9E, 0xD9, 0x78, 0x56, 0x79, 0xE0, 0x8A, 0xBC, 0x7C, 0x81, 0x16 }

N = 0xA9F593C09EAEEA8BF0C1CF6A

nonce[12] =

{ 0xA9, 0xF5, 0x93, 0xC0, 0x9E, 0xAE, 0xEA, 0x8B, 0xF0, 0xC1, 0xCF, 0x6A }

P=
0x0653B5714D1357F4995BDDACBE10873951A1EBA663718D1AF35D2F0D52C79DE49BE622C4A6
D90647BA2B004C3E8AE422FD27063AFA19AD883DCCBD97D98B8B0461B5671E75F19701C24042
B8D3AF79B9FF62BC448EF9440B1EA3F7E5C0F4BFEFE3E326E62D5EE4CB4B4CFFF30AD5F49A79
81ABF71617245B96E522E1ADD78A

pt[127] =

{ 0x06, 0x53, 0xB5, 0x71, 0x4D, 0x13, 0x57, 0xF4, 0x99, 0x5B, 0xDD, 0xAC, 0xBE, 0x10, 0x87, 0x39,

0x51, 0xA1, 0xEB, 0xA6, 0x63, 0x71, 0x8D, 0x1A, 0xF3, 0x5D, 0x2F, 0x0D, 0x52, 0xC7, 0x9D, 0xE4,

0x9B, 0xE6, 0x22, 0xC4, 0xA6, 0xD9, 0x06, 0x47, 0xBA, 0x2B, 0x00, 0x4C, 0x3E, 0x8A, 0xE4, 0x22,

0xFD, 0x27, 0x06, 0x3A, 0xFA, 0x19, 0xAD, 0x88, 0x3D, 0xCC, 0xBD, 0x97, 0xD9, 0x8B, 0x8B, 0x04,

0x61, 0xB5, 0x67, 0x1E, 0x75, 0xF1, 0x97, 0x01, 0xC2, 0x40, 0x42, 0xB8, 0xD3, 0xAF, 0x79, 0xB9,

0xFF, 0x62, 0xBC, 0x44, 0x8E, 0xF9, 0x44, 0x0B, 0x1E, 0xA3, 0xF7, 0xE5, 0xC0, 0xF4, 0xBF, 0xEF,

0xE3, 0xE3, 0x26, 0xE6, 0x2D, 0x5E, 0xE4, 0xCB, 0x4B, 0x4C, 0xFF, 0xF3, 0x0A, 0xD5, 0xF4, 0x9A,

0x79, 0x81, 0xAB, 0xF7, 0x16, 0x17, 0x24, 0x5B, 0x96, 0xE5, 0x22, 0xE1, 0xAD, 0xD7, 0x8A }

C_T=
0x5F82B9FCE34B94835395DD89D71FB758D2A3907FBF2FD58994A2B9CF8725AF26F0B23853C27A
06E35EE72CAD827713C18FA5DDA971D9BAA7B42A301FF60C6E4AD651C1BB6ED4F25F7D0FF38
7A11627934CD11F86984EA3AC969DDA9A020AD6424B0D393E3FB4B1119ADF5CDB012A59753E4
1D47E5E5A8C3A118ED407049B56D53BF56CB38C0B20A2502D1DA70B9761

c_t[143] =

{ 0x5F, 0x82, 0xB9, 0xFC, 0xE3, 0x4B, 0x94, 0x83, 0x53, 0x95, 0xDD, 0x89, 0xD7, 0x1F, 0xB7, 0x58,

0xD2, 0xA3, 0x90, 0x7F, 0xBF, 0x2F, 0xD5, 0x89, 0x94, 0xA2, 0xB9, 0xCF, 0x87, 0x25, 0xAF, 0x26,

0xF0, 0xB2, 0x38, 0x53, 0xC2, 0x7A, 0x06, 0xE3, 0x5E, 0xE7, 0x2C, 0xAD, 0x82, 0x77, 0x13, 0xC1,

0x8F, 0xA5, 0xDD, 0xA9, 0x71, 0xD9, 0xBA, 0xA7, 0xB4, 0x2A, 0x30, 0x1F, 0xF6, 0x0C, 0x6E, 0x4A,

0xD6, 0x51, 0xC1, 0xBB, 0x6E, 0xD4, 0xF2, 0x5F, 0x7D, 0x0F, 0xF3, 0x87, 0xA1, 0x16, 0x27, 0x93,

0x4C, 0xD1, 0x1F, 0x86, 0x98, 0x4E, 0xA3, 0xAC, 0x96, 0x9D, 0xDA, 0x9A, 0x02, 0x0A, 0xD6, 0x42,

0x4B, 0x0D, 0x39, 0x3E, 0x3F, 0xB4, 0xB1, 0x11, 0x9A, 0xDF, 0x5C, 0xDB, 0x01, 0x2A, 0x59, 0x75,

0x3E, 0x41, 0xD4, 0x7E, 0x5E, 0x5A, 0x8C, 0x3A, 0x11, 0x8E, 0xD4, 0x07, 0x04, 0x9B, 0x56, 0xD5,

0x3B, 0xF5, 0x6C, 0xB3, 0x8C, 0x0B, 0x20, 0xA2, 0x50, 0x2D, 0x1D, 0xA7, 0x0B, 0x97, 0x61 }

Test Vector #2:

K = 0xE58D5C8F8C9ED9785679E08ABC7C8116

key[16] =

{ 0xE5, 0x8D, 0x5C, 0x8F, 0x8C, 0x9E, 0xD9, 0x78, 0x56, 0x79, 0xE0, 0x8A, 0xBC, 0x7C, 0x81, 0x16 }

N = 0xA9F593C09EAEEA8BF0C1CF6A

nonce[12] =

{ 0xA9, 0xF5, 0x93, 0xC0, 0x9E, 0xAE, 0xEA, 0x8B, 0xF0, 0xC1, 0xCF, 0x6A }

P=
0xACA650CCCCDA604E16A8B54A3335E0BC2FD9444F33E3D9B82AFE6F445357634974F0F1728CF
113452321CBE5858304B01D4A14AE7F3B45980EE8033AD2A8599B78C29494C9E5F8945A8CADE3
EB5A30D156C0D83271626DADDB650954093443FBAC9701C02E5A973F39C2E1761A4B48C764BF6
DB215A54B285A06ECA3AF0A83F7

pt[128] =

{ 0xAC, 0xA6, 0x50, 0xCC, 0xCC, 0xDA, 0x60, 0x4E, 0x16, 0xA8, 0xB5, 0x4A, 0x33, 0x35, 0xE0, 0xBC,

0x2F, 0xD9, 0x44, 0x4F, 0x33, 0xE3, 0xD9, 0xB8, 0x2A, 0xFE, 0x6F, 0x44, 0x53, 0x57, 0x63, 0x49,

0x74, 0xF0, 0xF1, 0x72, 0x8C, 0xF1, 0x13, 0x45, 0x23, 0x21, 0xCB, 0xE5, 0x85, 0x83, 0x04, 0xB0,

0x1D, 0x4A, 0x14, 0xAE, 0x7F, 0x3B, 0x45, 0x98, 0x0E, 0xE8, 0x03, 0x3A, 0xD2, 0xA8, 0x59, 0x9B,

0x78, 0xC2, 0x94, 0x94, 0xC9, 0xE5, 0xF8, 0x94, 0x5A, 0x8C, 0xAD, 0xE3, 0xEB, 0x5A, 0x30, 0xD1,

0x56, 0xC0, 0xD8, 0x32, 0x71, 0x62, 0x6D, 0xAD, 0xDB, 0x65, 0x09, 0x54, 0x09, 0x34, 0x43, 0xFB,

0xAC, 0x97, 0x01, 0xC0, 0x2E, 0x5A, 0x97, 0x3F, 0x39, 0xC2, 0xE1, 0x76, 0x1A, 0x4B, 0x48, 0xC7,

0x64, 0xBF, 0x6D, 0xB2, 0x15, 0xA5, 0x4B, 0x28, 0x5A, 0x06, 0xEC, 0xA3, 0xAF, 0x0A, 0x83, 0xF7 }

C_T=
0xF5775C416282A339DC66B56F5A3AD0DDACDB3F96EFBD812B4D01F98686B5518B1FA4EBE5E8
5213E1C7EDE704397EF3536FC8CF3DF4FB52B7870E8EB2FD2FBCD5CF263231D2C09DCAE5C31C
DC99E36EFBE5737BF067D58A0A535B242BCBCA2A5604791E183CB0C2E5E851425E11B4E528237
F123B5DE8E349DD6D1A4506465F725700108000 3872271900D3F39C9661FD

c_t[144] =

{ 0xF5, 0x77, 0x5C, 0x41, 0x62, 0x82, 0xA3, 0x39, 0xDC, 0x66, 0xB5, 0x6F, 0x5A, 0x3A, 0xD0, 0xDD,

0xAC, 0xDB, 0x3F, 0x96, 0xEF, 0xBD, 0x81, 0x2B, 0x4D, 0x01, 0xF9, 0x86, 0x86, 0xB5, 0x51, 0x8B,

0x1F, 0xA4, 0xEB, 0xE5, 0xE8, 0x52, 0x13, 0xE1, 0xC7, 0xED, 0xE7, 0x04, 0x39, 0x7E, 0xF3, 0x53,

0x6F, 0xC8, 0xCF, 0x3D, 0xF4, 0xFB, 0x52, 0xB7, 0x87, 0x0E, 0x8E, 0xB2, 0xFD, 0x2F, 0xBC, 0xD5,

0xCF, 0x26, 0x32, 0x31, 0xD2, 0xC0, 0x9D, 0xCA, 0xE5, 0xC3, 0x1C, 0xDC, 0x99, 0xE3, 0x6E, 0xFB,

0xE5, 0x73, 0x7B, 0xF0, 0x67, 0xD5, 0x8A, 0x0A, 0x53, 0x5B, 0x24, 0x2B, 0xCB, 0xCA, 0x2A, 0x56,

0x04, 0x79, 0x1E, 0x18, 0x3C, 0xB0, 0xC2, 0xE5, 0xE8, 0x51, 0x42, 0x5E, 0x11, 0xB4, 0xE5, 0x28,

0x23, 0x7F, 0x12, 0x3B, 0x5D, 0xE8, 0xE3, 0x49, 0xDD, 0x6D, 0x1A, 0x45, 0x06, 0x46, 0x5F, 0x72,

0x57, 0x00, 0x10, 0x80, 0x00, 0x38, 0x72, 0x27, 0x19, 0x00, 0xD3, 0xF3, 0x9C, 0x96, 0x61, 0xFD }

Test Vector #3:

K = 0xE58D5C8F8C9ED9785679E08ABC7C8116

key[16] =

{ 0xE5, 0x8D, 0x5C, 0x8F, 0x8C, 0x9E, 0xD9, 0x78, 0x56, 0x79, 0xE0, 0x8A, 0xBC, 0x7C, 0x81, 0x16 }

N = 0xA9F593C09EAEEA8BF0C1CF6A

nonce[12] =

{ 0xA9, 0xF5, 0x93, 0xC0, 0x9E, 0xAE, 0xEA, 0x8B, 0xF0, 0xC1, 0xCF, 0x6A }

P=
0xD1AA8BBC04DFC92FFE2CB7748E70B02F5A91DA14781223A712D44C4BA14A1C78EB02387FE7
3FDCBCA8447056ACAA9B5F94D5208972B706DF9FC4C803EABB2BC58C3D8DF4AC496C34CB6B
AB939478CB417995B2314DAF7AF3F4C8A8D5D57A03F0EB2B7BBD2D16BABBF22C5B1EEBFF72
C7DD4F912D5821F9A6BFA2D063CE6F6648DF

pt[129] =

{ 0xD1, 0xAA, 0x8B, 0xBC, 0x04, 0xDF, 0xC9, 0x2F, 0xFE, 0x2C, 0xB7, 0x74, 0x8E, 0x70, 0xB0, 0x2F,

0x5A, 0x91, 0xDA, 0x14, 0x78, 0x12, 0x23, 0xA7, 0x12, 0xD4, 0x4C, 0x4B, 0xA1, 0x4A, 0x1C, 0x78,

0xEB, 0x02, 0x38, 0x7F, 0xE7, 0x3F, 0xDC, 0xBC, 0xA8, 0x44, 0x70, 0x56, 0xAC, 0xAA, 0x9B, 0x5F,

0x94, 0xD5, 0x20, 0x89, 0x72, 0xB7, 0x06, 0xDF, 0x9F, 0xC4, 0xC8, 0x03, 0xEA, 0xBB, 0x2B, 0xC5,

0x8C, 0x3D, 0x8D, 0xF4, 0xAC, 0x49, 0x6C, 0x34, 0xCB, 0x6B, 0xAB, 0x93, 0x94, 0x78, 0xCB, 0x41,

0x79, 0x95, 0xB2, 0x31, 0x4D, 0xAF, 0x7A, 0xF3, 0xF4, 0xC8, 0xA8, 0xD5, 0xD5, 0x7A, 0x03, 0xF0,

0xEB, 0x2B, 0x7B, 0xBD, 0x2D, 0x16, 0xBA, 0xBB, 0xF2, 0x2C, 0x5B, 0x1E, 0xEB, 0xFF, 0x72, 0xC7,

0xDD, 0x4F, 0x91, 0x2D, 0x58, 0x21, 0xF9, 0xA6, 0xBF, 0xA2, 0xD0, 0x63, 0xCE, 0x6F, 0x66, 0x48,

0xDF }

C_T=
0x887B8731AA870A5834E2B751E77F804ED993A1CDA44C7B34752BDA8974A82EBA805622E8839
CDC184C885CB710576CBCE657FB1AF97711F01622458BC53CCE8B3BD92B51B76C096A74241AA
CE6C1956BCA2611F35B189D547CF685AA17846A5D43C564653FFCEF6123BFF836E000DF289A8F
EEA4106C51C738C926856723BACDB3F5D0F87F7E29D94BF1B41DE8063E1071

c_t[145] =

{ 0x88, 0x7B, 0x87, 0x31, 0xAA, 0x87, 0x0A, 0x58, 0x34, 0xE2, 0xB7, 0x51, 0xE7, 0x7F, 0x80, 0x4E,

0xD9, 0x93, 0xA1, 0xCD, 0xA4, 0x4C, 0x7B, 0x34, 0x75, 0x2B, 0xDA, 0x89, 0x74, 0xA8, 0x2E, 0xBA,

0x80, 0x56, 0x22, 0xE8, 0x83, 0x9C, 0xDC, 0x18, 0x4C, 0x88, 0x5C, 0xB7, 0x10, 0x57, 0x6C, 0xBC,

0xE6, 0x57, 0xFB, 0x1A, 0xF9, 0x77, 0x11, 0xF0, 0x16, 0x22, 0x45, 0x8B, 0xC5, 0x3C, 0xCE, 0x8B,

0x3B, 0xD9, 0x2B, 0x51, 0xB7, 0x6C, 0x09, 0x6A, 0x74, 0x24, 0x1A, 0xAC, 0xE6, 0xC1, 0x95, 0x6B,

0xCA, 0x26, 0x11, 0xF3, 0x5B, 0x18, 0x9D, 0x54, 0x7C, 0xF6, 0x85, 0xAA, 0x17, 0x84, 0x6A, 0x5D,

0x43, 0xC5, 0x64, 0x65, 0x3F, 0xFC, 0xEF, 0x61, 0x23, 0xBF, 0xF8, 0x36, 0xE0, 0x00, 0xDF, 0x28,

0x9A, 0x8F, 0xEE, 0xA4, 0x10, 0x6C, 0x51, 0xC7, 0x38, 0xC9, 0x26, 0x85, 0x67, 0x23, 0xBA, 0xCD,

0xB3, 0xF5, 0xD0, 0xF8, 0x7F, 0x7E, 0x29, 0xD9, 0x4B, 0xF1, 0xB4, 0x1D, 0xE8, 0x06, 0x3E, 0x10,

0x71 }

Test Vector #4:

K = 0xB8453A728060F8D517BACEED3829F4D9

key[16] =

{ 0xB8, 0x45, 0x3A, 0x72, 0x80, 0x60, 0xF8, 0xD5, 0x17, 0xBA, 0xCE, 0xED, 0x38, 0x29, 0xF4, 0xD9 }

N = 0xCFBCE69C884D5BABBBAAF9A3

nonce[12] =

{ 0xCF, 0xBC, 0xE6, 0x9C, 0x88, 0x4D, 0x5B, 0xAB, 0xBB, 0xAA, 0xF9, 0xA3 }

P=
0xF7629B73DAE85A9BCA45C42EB7FC1818DC74A60E13AE65A043E24B5A4D3AE04C273E7D6F42
710F2D223D09EB7C1315718A5A1293D482E4C45C3E852E5106AAD7B695A02C4854801A5EFE937
A6540BCE8734E8141558C3433B1D4C733DC5EF9C47B5279AA46EE3D8BD33B0950BE5C9EBDF18
BCF069B6DAF82FF1186912F0ABA

pt[127] =

{ 0xF7, 0x62, 0x9B, 0x73, 0xDA, 0xE8, 0x5A, 0x9B, 0xCA, 0x45, 0xC4, 0x2E, 0xB7, 0xFC, 0x18, 0x18,

0xDC, 0x74, 0xA6, 0x0E, 0x13, 0xAE, 0x65, 0xA0, 0x43, 0xE2, 0x4B, 0x5A, 0x4D, 0x3A, 0xE0, 0x4C,

0x27, 0x3E, 0x7D, 0x6F, 0x42, 0x71, 0x0F, 0x2D, 0x22, 0x3D, 0x09, 0xEB, 0x7C, 0x13, 0x15, 0x71,

0x8A, 0x5A, 0x12, 0x93, 0xD4, 0x82, 0xE4, 0xC4, 0x5C, 0x3E, 0x85, 0x2E, 0x51, 0x06, 0xAA, 0xD7,

0xB6, 0x95, 0xA0, 0x2C, 0x48, 0x54, 0x80, 0x1A, 0x5E, 0xFE, 0x93, 0x7A, 0x65, 0x40, 0xBC, 0xE8,

0x73, 0x4E, 0x81, 0x41, 0x55, 0x8C, 0x34, 0x33, 0xB1, 0xD4, 0xC7, 0x33, 0xDC, 0x5E, 0xF9, 0xC4,

0x7B, 0x52, 0x79, 0xAA, 0x46, 0xEE, 0x3D, 0x8B, 0xD3, 0x3B, 0x09, 0x50, 0xBE, 0x5C, 0x9E, 0xBD,

0xF1, 0x8B, 0xCF, 0x06, 0x9B, 0x6D, 0xAF, 0x82, 0xFF, 0x11, 0x86, 0x91, 0x2F, 0x0A, 0xBA }

C_T=
0xDEDE575B6EFE390F2CBB4F368A711F6CDF69ABD11AF580B2BF4029F85EB835D1ABDDB30E9
E9CF3F13CBA3BCC2E918713D218AF0D07CC614AF69892AFA986AF2D5E60EDB05D09D3B29E2A

65B543AD6F26E5D76B660FE9184906A6315CD6B5355FA291A1E90C510DF20E46C116E2180009C2
87659DB8D45CC3968049FA29F08DE5D156EDF7B0DBC84E410F292868C4BE

c_t[143] =

{ 0xDE, 0xDE, 0x57, 0x5B, 0x6E, 0xFE, 0x39, 0x0F, 0x2C, 0xBB, 0x4F, 0x36, 0x8A, 0x71, 0x1F, 0x6C,

0xDF, 0x69, 0xAB, 0xD1, 0x1A, 0xF5, 0x80, 0xB2, 0xBF, 0x40, 0x29, 0xF8, 0x5E, 0xB8, 0x35, 0xD1,

0xAB, 0xDD, 0xB3, 0x0E, 0x9E, 0x9C, 0xF3, 0xF1, 0x3C, 0xBA, 0x3B, 0xCC, 0x2E, 0x91, 0x87, 0x13,

0xD2, 0x18, 0xAF, 0x0D, 0x07, 0xCC, 0x61, 0x4A, 0xF6, 0x98, 0x92, 0xAF, 0xA9, 0x86, 0xAF, 0x2D,

0x5E, 0x60, 0xED, 0xB0, 0x5D, 0x09, 0xD3, 0xB2, 0x9E, 0x2A, 0x65, 0xB5, 0x43, 0xAD, 0x6F, 0x26,

0xE5, 0xD7, 0x6B, 0x66, 0x0F, 0xE9, 0x18, 0x49, 0x06, 0xA6, 0x31, 0x5C, 0xD6, 0xB5, 0x35, 0x5F,

0xA2, 0x91, 0xA1, 0xE9, 0x0C, 0x51, 0x0D, 0xF2, 0x0E, 0x46, 0xC1, 0x16, 0xE2, 0x18, 0x00, 0x09,

0xC2, 0x87, 0x65, 0x9D, 0xB8, 0xD4, 0x5C, 0xC3, 0x96, 0x80, 0x49, 0xFA, 0x29, 0xF0, 0x8D, 0xE5,

0xD1, 0x56, 0xED, 0xF7, 0xB0, 0xDB, 0xC8, 0x4E, 0x41, 0x0F, 0x29, 0x28, 0x68, 0xC4, 0xBE }

Test Vector #5:

K = 0xB8453A728060F8D517BACEED3829F4D9

key[16] = { 0xB8, 0x45, 0x3A, 0x72, 0x80, 0x60, 0xF8, 0xD5, 0x17, 0xBA, 0xCE, 0xED, 0x38, 0x29,
0xF4, 0xD9 }

N = 0xCFBCE69C884D5BABBBAAF9A3

nonce[12] = { 0xCF, 0xBC, 0xE6, 0x9C, 0x88, 0x4D, 0x5B, 0xAB, 0xBB, 0xAA, 0xF9, 0xA3 }

P=
0x29B4013F552FBCE993544CC6605CB05C62A7894C4C99E6A12C5F9F2EE4DFBEBAD70CDD0893
542240F28BB5FBB9090332ED110ABFAE6C4C6460D916F8994136575B5A6FD8DB605FDF14CB819
77AFF7F99B5272580BF220133C691B09BADC4D1FE7125FD17FDBFC103E3F00A4D8E5A6F1E3D3
AF2A908535DE858E1CCD3DB4D1835

pt[128] =

{ 0x29, 0xB4, 0x01, 0x3F, 0x55, 0x2F, 0xBC, 0xE9, 0x93, 0x54, 0x4C, 0xC6, 0x60, 0x5C, 0xB0, 0x5C,

0x62, 0xA7, 0x89, 0x4C, 0x4C, 0x99, 0xE6, 0xA1, 0x2C, 0x5F, 0x9F, 0x2E, 0xE4, 0xDF, 0xBE, 0xBA,

0xD7, 0x0C, 0xDD, 0x08, 0x93, 0x54, 0x22, 0x40, 0xF2, 0x8B, 0xB5, 0xFB, 0xB9, 0x09, 0x03, 0x32,

0xED, 0x11, 0x0A, 0xBF, 0xAE, 0x6C, 0x4C, 0x64, 0x60, 0xD9, 0x16, 0xF8, 0x99, 0x41, 0x36, 0x57,

0x5B, 0x5A, 0x6F, 0xD8, 0xDB, 0x60, 0x5F, 0xDF, 0x14, 0xCB, 0x81, 0x97, 0x7A, 0xFF, 0x7F, 0x99,

0xB5, 0x27, 0x25, 0x80, 0xBF, 0x22, 0x01, 0x33, 0xC6, 0x91, 0xB0, 0x9B, 0xAD, 0xC4, 0xD1, 0xFE,

0x71, 0x25, 0xFD, 0x17, 0xFD, 0xBF, 0xC1, 0x03, 0xE3, 0xF0, 0x0A, 0x4D, 0x8E, 0x5A, 0x6F, 0x1E,

0x3D, 0x3A, 0xF2, 0xA9, 0x08, 0x53, 0x5D, 0xE8, 0x58, 0xE1, 0xCC, 0xD3, 0xDB, 0x4D, 0x18, 0x35 }

C_T=
0x0008CD17E139DF7D75AAC7DE5DD1B72861BA849345C203B3D0FDFD8CF75D6B275BEF13694F
B9DE9CEC0C87DCEB8B9150B553B7217D22C9EACA7F017961C133ADB3AF2244CE3D0C77D41F7
7585C12AC5723BECFA7E5472D4971E346F4A72F1D65A8E62554B700F17A3E8DC20BD21EF1AA0E
3658322BEAAEA9317003B8DDB72FFDFA0834974152B95BADE2DF83D7EEC455

c_t[144] =

{ 0x00, 0x08, 0xCD, 0x17, 0xE1, 0x39, 0xDF, 0x7D, 0x75, 0xAA, 0xC7, 0xDE, 0x5D, 0xD1, 0xB7, 0x28,

0x61, 0xBA, 0x84, 0x93, 0x45, 0xC2, 0x03, 0xB3, 0xD0, 0xFD, 0xFD, 0x8C, 0xF7, 0x5D, 0x6B, 0x27,

0x5B, 0xEF, 0x13, 0x69, 0x4F, 0xB9, 0xDE, 0x9C, 0xEC, 0x0C, 0x87, 0xDC, 0xEB, 0x8B, 0x91, 0x50,

0xB5, 0x53, 0xB7, 0x21, 0x7D, 0x22, 0xC9, 0xEA, 0xCA, 0x7F, 0x01, 0x79, 0x61, 0xC1, 0x33, 0xAD,

0xB3, 0xAF, 0x22, 0x44, 0xCE, 0x3D, 0x0C, 0x77, 0xD4, 0x1F, 0x77, 0x58, 0x5C, 0x12, 0xAC, 0x57,

0x23, 0xBE, 0xCF, 0xA7, 0xE5, 0x47, 0x2D, 0x49, 0x71, 0xE3, 0x46, 0xF4, 0xA7, 0x2F, 0x1D, 0x65,

0xA8, 0xE6, 0x25, 0x54, 0xB7, 0x00, 0xF1, 0x7A, 0x3E, 0x8D, 0xC2, 0x0B, 0xD2, 0x1E, 0xF1, 0xAA,

0x0E, 0x36, 0x58, 0x32, 0x2B, 0xEA, 0xAE, 0xA9, 0x31, 0x70, 0x03, 0xB8, 0xDD, 0xB7, 0x2F, 0xFD,

0xFA, 0x08, 0x34, 0x97, 0x41, 0x52, 0xB9, 0x5B, 0xAD, 0xE2, 0xDF, 0x83, 0xD7, 0xEE, 0xC4, 0x55 }

Test Vector #6:

K = 0xB8453A728060F8D517BACEED3829F4D9

key[16] =

{ 0xB8, 0x45, 0x3A, 0x72, 0x80, 0x60, 0xF8, 0xD5, 0x17, 0xBA, 0xCE, 0xED, 0x38, 0x29, 0xF4, 0xD9 }

N = 0xCFBCE69C884D5BABBBAAF9A3

nonce[12] =

{ 0xCF, 0xBC, 0xE6, 0x9C, 0x88, 0x4D, 0x5B, 0xAB, 0xBB, 0xAA, 0xF9, 0xA3 }

P=
0x1D76BDF0626A7134BEB28A90D54ED7796C4C9535465C090C4B583A8CD40EF0A3864E7C07CC
AED140DF6B9D73234E652F8FF425FC206F63DFAB7DCDBBBE30411A14695E72A2BD8C4BFB1D6
991DB4F99EEA7435E55261E37FDF57CE79DF725C810192F5E6E0331ED62EB8A72C5B9DA6DFD97
48B3D168A69BAB33319EFD1E84EF2570

pt[129] =

{ 0x1D, 0x76, 0xBD, 0xF0, 0x62, 0x6A, 0x71, 0x34, 0xBE, 0xB2, 0x8A, 0x90, 0xD5, 0x4E, 0xD7, 0x79,
0x6C, 0x4C, 0x95, 0x35, 0x46, 0x5C, 0x09, 0x0C, 0x4B, 0x58, 0x3A, 0x8C, 0xD4, 0x0E, 0xF0, 0xA3,
0x86, 0x4E, 0x7C, 0x07, 0xCC, 0xAE, 0xD1, 0x40, 0xDF, 0x6B, 0x9D, 0x73, 0x23, 0x4E, 0x65, 0x2F,
0x8F, 0xF4, 0x25, 0xFC, 0x20, 0x6F, 0x63, 0xDF, 0xAB, 0x7D, 0xCD, 0xBB, 0xBE, 0x30, 0x41, 0x1A,
0x14, 0x69, 0x5E, 0x72, 0xA2, 0xBD, 0x8C, 0x4B, 0xFB, 0x1D, 0x69, 0x91, 0xDB, 0x4F, 0x99, 0xEE,
0xA7, 0x43, 0x5E, 0x55, 0x26, 0x1E, 0x37, 0xFD, 0xF5, 0x7C, 0xE7, 0x9D, 0xF7, 0x25, 0xC8, 0x10,
0x19, 0x2F, 0x5E, 0x6E, 0x03, 0x31, 0xED, 0x62, 0xEB, 0x8A, 0x72, 0xC5, 0xB9, 0xDA, 0x6D, 0xFD,
0x97, 0x48, 0xB3, 0xD1, 0x68, 0xA6, 0x9B, 0xAB, 0x33, 0x31, 0x9E, 0xFD, 0x1E, 0x84, 0xEF, 0x25,
0x70 }

C_T=
0x34CA71D8D67C12A0584C0188E8C3D00D6F5198EA4F07EC1EB7FA582EC78C253E0AADB26610
432D9CC1ECAF5471CCF74DD7B69862F321E65101DBDA3A46B044E0FC9C13EEB7E0DFE33BC99
F5EFDA24A2031DAB4727C7B1B87420E11F2FDCE048BC0EC862D498EDD1B36F7BA83E59EF349
A444194A4B1F68EA5AA05196187ED8ED684826C0C356A9B8EDA55BD91C2BA1022B

c_t[145] =

{ 0x34, 0xCA, 0x71, 0xD8, 0xD6, 0x7C, 0x12, 0xA0, 0x58, 0x4C, 0x01, 0x88, 0xE8, 0xC3, 0xD0, 0x0D,
0x6F, 0x51, 0x98, 0xEA, 0x4F, 0x07, 0xEC, 0x1E, 0xB7, 0xFA, 0x58, 0x2E, 0xC7, 0x8C, 0x25, 0x3E,
0x0A, 0xAD, 0xB2, 0x66, 0x10, 0x43, 0x2D, 0x9C, 0xC1, 0xEC, 0xAF, 0x54, 0x71, 0xCC, 0xF7, 0x4D,
0xD7, 0xB6, 0x98, 0x62, 0xF3, 0x21, 0xE6, 0x51, 0x01, 0xDB, 0xDA, 0x3A, 0x46, 0xB0, 0x44, 0xE0,
0xFC, 0x9C, 0x13, 0xEE, 0xB7, 0xE0, 0xDF, 0xE3, 0x3B, 0xC9, 0x9F, 0x5E, 0xFD, 0xA2, 0x4A, 0x20,
0x31, 0xDA, 0xB4, 0x72, 0x7C, 0x7B, 0x1B, 0x87, 0x42, 0x0E, 0x11, 0xF2, 0xFD, 0xCE, 0x04, 0x8B,
0xC0, 0xEC, 0x86, 0x2D, 0x49, 0x8E, 0xDD, 0x1B, 0x36, 0xF7, 0xBA, 0x83, 0xE5, 0x9E, 0xF3, 0x49,
0xA4, 0x44, 0x19, 0x4A, 0x4B, 0x1F, 0x68, 0xEA, 0x5A, 0xA0, 0x51, 0x96, 0x18, 0x7E, 0xD8, 0xED,
0x68, 0x48, 0x26, 0xC0, 0xC3, 0x56, 0xA9, 0xB8, 0xED, 0xA5, 0x5B, 0xD9, 0x1C, 0x2B, 0xA1, 0x02,
0x2B }

D.6.2 ECIES

==========

ECIES Encryption as per 1609.2,

Used to wrap AES-CCM 128-bit keys

Encryption Inputs:

- R: {ec256 point} Recipient public key

- k: {octet string} AES-CCM 128-bit key to be wrapped (128 bits)

- P1: {octet string} SHA-256 hash of some defined recipient info or of an empty string (256 bits)

Encryption Outputs:

- V: {ec256 point} Sender's ephemeral public key

- C: {octet string} Ciphertext, i.e. enc(k) (128 bits)

- T: {octet string} Authentication tag, (128 bits)

The encryption output is randomised, due to the ephemeral sender's key (v,V)

In the script, for testing purpose:

- v is an optional input to ecies_enc()

- v is an output of ecies_enc() to be printed in the test vectors

Test Vector #1:

==========

Sender's ephemeral private key:

v = 0x1384C31D6982D52BCA3BED8A7E60F52FECDAB44E5C0EA166815A8159E09FFB42

v[32] =

{ 0x13, 0x84, 0xC3, 0x1D, 0x69, 0x82, 0xD5, 0x2B, 0xCA, 0x3B, 0xED, 0x8A, 0x7E, 0x60, 0xF5, 0x2F,

0xEC, 0xDA, 0xB4, 0x4E, 0x5C, 0x0E, 0xA1, 0x66, 0x81, 0x5A, 0x81, 0x59, 0xE0, 0x9F, 0xFB, 0x42 }

AES key to be encrypted (wrapped):

k = 0x9169155B08B07674CBADF75FB46A7B0D

k[16] =

{ 0x91, 0x69, 0x15, 0x5B, 0x08, 0xB0, 0x76, 0x74, 0xCB, 0xAD, 0xF7, 0x5F, 0xB4, 0x6A, 0x7B, 0x0D }

Hash(RecipientInfo):

P1 = 0x9169155B08B07674CBADF75FB46A7B0D

P1[16] =

{ 0x91, 0x69, 0x15, 0x5B, 0x08, 0xB0, 0x76, 0x74, 0xCB, 0xAD, 0xF7, 0x5F, 0xB4, 0x6A, 0x7B, 0x0D }

Recipient's private key (Decryption input):

r = 0x060E41440A4E35154CA0EFCB52412145836AD032833E6BC781E533BF14851085

r[32] =

{ 0x06, 0x0E, 0x41, 0x44, 0x0A, 0x4E, 0x35, 0x15, 0x4C, 0xA0, 0xEF, 0xCB, 0x52, 0x41, 0x21, 0x45,
0x83, 0x6A, 0xD0, 0x32, 0x83, 0x3E, 0x6B, 0xC7, 0x81, 0xE5, 0x33, 0xBF, 0x14, 0x85, 0x10, 0x85 }

Recipient's public key (x-coordinate):

Rx = 0x8C5E20FE31935F6FA682A1F6D46E4468534FFEA1A698B14B0B12513EED8DEB11

Rx[32] =

{ 0x8C, 0x5E, 0x20, 0xFE, 0x31, 0x93, 0x5F, 0x6F, 0xA6, 0x82, 0xA1, 0xF6, 0xD4, 0x6E, 0x44, 0x68,
0x53, 0x4F, 0xFE, 0xA1, 0xA6, 0x98, 0xB1, 0x4B, 0x0B, 0x12, 0x51, 0x3E, 0xED, 0x8D, 0xEB, 0x11 }

Recipient's public key (y-coordinate):

Ry = 0x1270FEC2427E6A154DFCAE3368584396C8251A04E2AE7D87B016FF65D22D6F9E

Ry[32] =

{ 0x12, 0x70, 0xFE, 0xC2, 0x42, 0x7E, 0x6A, 0x15, 0x4D, 0xFC, 0xAE, 0x33, 0x68, 0x58, 0x43, 0x96,
0xC8, 0x25, 0x1A, 0x04, 0xE2, 0xAE, 0x7D, 0x87, 0xB0, 0x16, 0xFF, 0x65, 0xD2, 0x2D, 0x6F, 0x9E }

Encryption Output:

Sender's ephemeral public key (x-coordinate):

Vx = 0xF45A99137B1BB2C150D6D8CF7292CA07DA68C003DAA766A9AF7F67F5EE916828

Vx[32] =

{ 0xF4, 0x5A, 0x99, 0x13, 0x7B, 0x1B, 0xB2, 0xC1, 0x50, 0xD6, 0xD8, 0xCF, 0x72, 0x92, 0xCA, 0x07,

0xDA, 0x68, 0xC0, 0x03, 0xDA, 0xA7, 0x66, 0xA9, 0xAF, 0x7F, 0x67, 0xF5, 0xEE, 0x91, 0x68, 0x28 }

Sender's ephemeral public key (y-coordinate):

Vy = 0xF6A25216F44CB64A96C229AE00B479857B3B81C1319FB2ADF0E8DB2681769729

Vx[32] =

{ 0xF6, 0xA2, 0x52, 0x16, 0xF4, 0x4C, 0xB6, 0x4A, 0x96, 0xC2, 0x29, 0xAE, 0x00, 0xB4, 0x79, 0x85,

0x7B, 0x3B, 0x81, 0xC1, 0x31, 0x9F, 0xB2, 0xAD, 0xF0, 0xE8, 0xDB, 0x26, 0x81, 0x76, 0x97, 0x29 }

Encrypted (wrapped) AES key:

C = 0xA6342013D623AD6C5F6882469673AE33

C[16] =

{ 0xA6, 0x34, 0x20, 0x13, 0xD6, 0x23, 0xAD, 0x6C, 0x5F, 0x68, 0x82, 0x46, 0x96, 0x73, 0xAE, 0x33 }

Authentication tag:

T = 0x80e1d85d30f1bae4ecf1a534a89a0786

T[16] =

{ 0x80, 0xE1, 0xD8, 0x5D, 0x30, 0xF1, 0xBA, 0xE4, 0xEC, 0xF1, 0xA5, 0x34, 0xA8, 0x9A, 0x07, 0x86 }

Test Vector #2:

==========

Sender's ephemeral private key:

v = 0xD418760F0CB2DCB856BC3C7217AD3AA36DB6742AE1DB655A3D28DF88CBBF84E1

v[32] =

{ 0xD4, 0x18, 0x76, 0x0F, 0x0C, 0xB2, 0xDC, 0xB8, 0x56, 0xBC, 0x3C, 0x72, 0x17, 0xAD, 0x3A, 0xA3,

0x6D, 0xB6, 0x74, 0x2A, 0xE1, 0xDB, 0x65, 0x5A, 0x3D, 0x28, 0xDF, 0x88, 0xCB, 0xBF, 0x84, 0xE1 }

AES key to be encrypted (wrapped):

k = 0x9169155B08B07674CBADF75FB46A7B0D

k[16] =

{ 0x91, 0x69, 0x15, 0x5B, 0x08, 0xB0, 0x76, 0x74, 0xCB, 0xAD, 0xF7, 0x5F, 0xB4, 0x6A, 0x7B, 0x0D }

Hash(RecipientInfo):

P1 = 0x9169155B08B07674CBADF75FB46A7B0D

P1[16] =

{ 0x91, 0x69, 0x15, 0x5B, 0x08, 0xB0, 0x76, 0x74, 0xCB, 0xAD, 0xF7, 0x5F, 0xB4, 0x6A, 0x7B, 0x0D }

Recipient's private key (Decryption input):

r = 0x060E41440A4E35154CA0EFCB52412145836AD032833E6BC781E533BF14851085

r[32] =

{ 0x06, 0x0E, 0x41, 0x44, 0x0A, 0x4E, 0x35, 0x15, 0x4C, 0xA0, 0xEF, 0xCB, 0x52, 0x41, 0x21, 0x45,
0x83, 0x6A, 0xD0, 0x32, 0x83, 0x3E, 0x6B, 0xC7, 0x81, 0xE5, 0x33, 0xBF, 0x14, 0x85, 0x10, 0x85 }

Recipient's public key (x-coordinate):

Rx = 0x8C5E20FE31935F6FA682A1F6D46E4468534FFEA1A698B14B0B12513EED8DEB11

Rx[32] =

{ 0x8C, 0x5E, 0x20, 0xFE, 0x31, 0x93, 0x5F, 0x6F, 0xA6, 0x82, 0xA1, 0xF6, 0xD4, 0x6E, 0x44, 0x68,
0x53, 0x4F, 0xFE, 0xA1, 0xA6, 0x98, 0xB1, 0x4B, 0x0B, 0x12, 0x51, 0x3E, 0xED, 0x8D, 0xEB, 0x11 }

Recipient's public key (y-coordinate):

Ry = 0x1270FEC2427E6A154DFCAE3368584396C8251A04E2AE7D87B016FF65D22D6F9E

Ry[32] =

{ 0x12, 0x70, 0xFE, 0xC2, 0x42, 0x7E, 0x6A, 0x15, 0x4D, 0xFC, 0xAE, 0x33, 0x68, 0x58, 0x43, 0x96,
0xC8, 0x25, 0x1A, 0x04, 0xE2, 0xAE, 0x7D, 0x87, 0xB0, 0x16, 0xFF, 0x65, 0xD2, 0x2D, 0x6F, 0x9E }

Encryption Output:

Sender's ephemeral public key (x-coordinate):

Vx = 0xEE9CC7FBD9EDECEA41F7C8BD258E8D2E988E75BD069ADDCA1E5A38E534AC6818

Vx[32] =

{ 0xEE, 0x9C, 0xC7, 0xFB, 0xD9, 0xED, 0xEC, 0xEA, 0x41, 0xF7, 0xC8, 0xBD, 0x25, 0x8E, 0x8D, 0x2E,
0x98, 0x8E, 0x75, 0xBD, 0x06, 0x9A, 0xDD, 0xCA, 0x1E, 0x5A, 0x38, 0xE5, 0x34, 0xAC, 0x68, 0x18 }

Sender's ephemeral public key (y-coordinate):

Vy = 0x5AE3C8D9FE0B1FC7438F29417C240F8BF81C358EC1A4D0C6E98D8EDBCC714017

Vx[32] =

{ 0x5A, 0xE3, 0xC8, 0xD9, 0xFE, 0x0B, 0x1F, 0xC7, 0x43, 0x8F, 0x29, 0x41, 0x7C, 0x24, 0x0F, 0x8B,
0xF8, 0x1C, 0x35, 0x8E, 0xC1, 0xA4, 0xD0, 0xC6, 0xE9, 0x8D, 0x8E, 0xDB, 0xCC, 0x71, 0x40, 0x17 }

Encrypted (wrapped) AES key:

C = 0xDD530BE3BCD149E881E09F06E160F5A0

C[16] =

{ 0xDD, 0x53, 0x0B, 0xE3, 0xBC, 0xD1, 0x49, 0xE8, 0x81, 0xE0, 0x9F, 0x06, 0xE1, 0x60, 0xF5, 0xA0 }

Authentication tag:

T = 0x06c1f0f5eaed453caf78e01a3d16a001

T[16] =

{ 0x06, 0xC1, 0xF0, 0xF5, 0xEA, 0xED, 0x45, 0x3C, 0xAF, 0x78, 0xE0, 0x1A, 0x3D, 0x16, 0xA0, 0x01 }

Test Vector #3:

==========

Sender's ephemeral private key:

v = 0x1384C31D6982D52BCA3BED8A7E60F52FECDAB44E5C0EA166815A8159E09FFB42

v[32] =

{ 0x13, 0x84, 0xC3, 0x1D, 0x69, 0x82, 0xD5, 0x2B, 0xCA, 0x3B, 0xED, 0x8A, 0x7E, 0x60, 0xF5, 0x2F,
0xEC, 0xDA, 0xB4, 0x4E, 0x5C, 0x0E, 0xA1, 0x66, 0x81, 0x5A, 0x81, 0x59, 0xE0, 0x9F, 0xFB, 0x42 }

AES key to be encrypted (wrapped):

k = 0x687E9757DEBFD87B0C267330C183C7B6

k[16] =

{ 0x68, 0x7E, 0x97, 0x57, 0xDE, 0xBF, 0xD8, 0x7B, 0x0C, 0x26, 0x73, 0x30, 0xC1, 0x83, 0xC7, 0xB6 }

Hash(RecipientInfo):

P1 = 0x687E9757DEBFD87B0C267330C183C7B6

P1[16] =

{ 0x68, 0x7E, 0x97, 0x57, 0xDE, 0xBF, 0xD8, 0x7B, 0x0C, 0x26, 0x73, 0x30, 0xC1, 0x83, 0xC7, 0xB6 }

Recipient's private key (Decryption input):

r = 0xDA5E1D853FCC5D0C162A245B9F29D38EB6059F0DB172FB7FDA6663B925E8C744

r[32] =

{ 0xDA, 0x5E, 0x1D, 0x85, 0x3F, 0xCC, 0x5D, 0x0C, 0x16, 0x2A, 0x24, 0x5B, 0x9F, 0x29, 0xD3, 0x8E,
0xB6, 0x05, 0x9F, 0x0D, 0xB1, 0x72, 0xFB, 0x7F, 0xDA, 0x66, 0x63, 0xB9, 0x25, 0xE8, 0xC7, 0x44 }

Recipient's public key (x-coordinate):

Rx = 0x8008B06FC4C9F9856048DA186E7DC390963D6A424E80B274FB75D12188D7D73F

Rx[32] =

{ 0x80, 0x08, 0xB0, 0x6F, 0xC4, 0xC9, 0xF9, 0x85, 0x60, 0x48, 0xDA, 0x18, 0x6E, 0x7D, 0xC3, 0x90,
0x96, 0x3D, 0x6A, 0x42, 0x4E, 0x80, 0xB2, 0x74, 0xFB, 0x75, 0xD1, 0x21, 0x88, 0xD7, 0xD7, 0x3F }

Recipient's public key (y-coordinate):

Ry = 0x2774FB9600F27D7B3BBB2F7FCD8D2C96D4619EF9B4692C6A7C5733B5BAC8B27D

Ry[32] =

{ 0x27, 0x74, 0xFB, 0x96, 0x00, 0xF2, 0x7D, 0x7B, 0x3B, 0xBB, 0x2F, 0x7F, 0xCD, 0x8D, 0x2C, 0x96,

0xD4, 0x61, 0x9E, 0xF9, 0xB4, 0x69, 0x2C, 0x6A, 0x7C, 0x57, 0x33, 0xB5, 0xBA, 0xC8, 0xB2, 0x7D }

Encryption Output:

Sender's ephemeral public key (x-coordinate):

Vx = 0xF45A99137B1BB2C150D6D8CF7292CA07DA68C003DAA766A9AF7F67F5EE916828

Vx[32] =

{ 0xF4, 0x5A, 0x99, 0x13, 0x7B, 0x1B, 0xB2, 0xC1, 0x50, 0xD6, 0xD8, 0xCF, 0x72, 0x92, 0xCA, 0x07,

0xDA, 0x68, 0xC0, 0x03, 0xDA, 0xA7, 0x66, 0xA9, 0xAF, 0x7F, 0x67, 0xF5, 0xEE, 0x91, 0x68, 0x28 }

Sender's ephemeral public key (y-coordinate):

Vy = 0xF6A25216F44CB64A96C229AE00B479857B3B81C1319FB2ADF0E8DB2681769729

Vx[32] =

{ 0xF6, 0xA2, 0x52, 0x16, 0xF4, 0x4C, 0xB6, 0x4A, 0x96, 0xC2, 0x29, 0xAE, 0x00, 0xB4, 0x79, 0x85,

0x7B, 0x3B, 0x81, 0xC1, 0x31, 0x9F, 0xB2, 0xAD, 0xF0, 0xE8, 0xDB, 0x26, 0x81, 0x76, 0x97, 0x29 }

Encrypted (wrapped) AES key:

C = 0x1F6346EDAEAF57561FC9604FEBEFF44E

C[16] =

{ 0x1F, 0x63, 0x46, 0xED, 0xAE, 0xAF, 0x57, 0x56, 0x1F, 0xC9, 0x60, 0x4F, 0xEB, 0xEF, 0xF4, 0x4E }

Authentication tag:

T = 0x373c0fa7c52a0798ec36eadfe387c3ef

T[16] =

{ 0x37, 0x3C, 0x0F, 0xA7, 0xC5, 0x2A, 0x07, 0x98, 0xEC, 0x36, 0xEA, 0xDF, 0xE3, 0x87, 0xC3, 0xEF }

Test Vector #4:

==========

Sender's ephemeral private key:

v = 0x4624A6F9F6BC6BD088A71ED97B3AEE983B5CC2F574F64E96A531D2464137049F

v[32] =

{ 0x46, 0x24, 0xA6, 0xF9, 0xF6, 0xBC, 0x6B, 0xD0, 0x88, 0xA7, 0x1E, 0xD9, 0x7B, 0x3A, 0xEE, 0x98,

0x3B, 0x5C, 0xC2, 0xF5, 0x74, 0xF6, 0x4E, 0x96, 0xA5, 0x31, 0xD2, 0x46, 0x41, 0x37, 0x04, 0x9F }

AES key to be encrypted (wrapped):

k = 0x687E9757DEBFD87B0C267330C183C7B6

k[16] =

{ 0x68, 0x7E, 0x97, 0x57, 0xDE, 0xBF, 0xD8, 0x7B, 0x0C, 0x26, 0x73, 0x30, 0xC1, 0x83, 0xC7, 0xB6 }

Hash(RecipientInfo):

P1 = 0x687E9757DEBFD87B0C267330C183C7B6

P1[16] =

{ 0x68, 0x7E, 0x97, 0x57, 0xDE, 0xBF, 0xD8, 0x7B, 0x0C, 0x26, 0x73, 0x30, 0xC1, 0x83, 0xC7, 0xB6 }

Recipient's private key (Decryption input):

r = 0xDA5E1D853FCC5D0C162A245B9F29D38EB6059F0DB172FB7FDA6663B925E8C744

r[32] =

{ 0xDA, 0x5E, 0x1D, 0x85, 0x3F, 0xCC, 0x5D, 0x0C, 0x16, 0x2A, 0x24, 0x5B, 0x9F, 0x29, 0xD3, 0x8E,

0xB6, 0x05, 0x9F, 0x0D, 0xB1, 0x72, 0xFB, 0x7F, 0xDA, 0x66, 0x63, 0xB9, 0x25, 0xE8, 0xC7, 0x44 }

Recipient's public key (x-coordinate):

Rx = 0x8008B06FC4C9F9856048DA186E7DC390963D6A424E80B274FB75D12188D7D73F

Rx[32] =

{ 0x80, 0x08, 0xB0, 0x6F, 0xC4, 0xC9, 0xF9, 0x85, 0x60, 0x48, 0xDA, 0x18, 0x6E, 0x7D, 0xC3, 0x90,

0x96, 0x3D, 0x6A, 0x42, 0x4E, 0x80, 0xB2, 0x74, 0xFB, 0x75, 0xD1, 0x21, 0x88, 0xD7, 0xD7, 0x3F }

Recipient's public key (y-coordinate):

Ry = 0x2774FB9600F27D7B3BBB2F7FCD8D2C96D4619EF9B4692C6A7C5733B5BAC8B27D

Ry[32] =

{ 0x27, 0x74, 0xFB, 0x96, 0x00, 0xF2, 0x7D, 0x7B, 0x3B, 0xBB, 0x2F, 0x7F, 0xCD, 0x8D, 0x2C, 0x96,

0xD4, 0x61, 0x9E, 0xF9, 0xB4, 0x69, 0x2C, 0x6A, 0x7C, 0x57, 0x33, 0xB5, 0xBA, 0xC8, 0xB2, 0x7D }

Encryption Output:

Sender's ephemeral public key (x-coordinate):

Vx = 0x121AA495C6B2C07A2B2DAEC36BD207D6620D7E6081050DF5DE3E9696868FCDCA

Vx[32] =

{ 0x12, 0x1A, 0xA4, 0x95, 0xC6, 0xB2, 0xC0, 0x7A, 0x2B, 0x2D, 0xAE, 0xC3, 0x6B, 0xD2, 0x07, 0xD6,

0x62, 0x0D, 0x7E, 0x60, 0x81, 0x05, 0x0D, 0xF5, 0xDE, 0x3E, 0x96, 0x96, 0x86, 0x8F, 0xCD, 0xCA }

Sender's ephemeral public key (y-coordinate):

Vy = 0x46C31A1ABEA0BDDAAAAEFBBA3AFDBFF1AC8D196BC313FC130926810C05503950

Vx[32] =

{ 0x46, 0xC3, 0x1A, 0x1A, 0xBE, 0xA0, 0xBD, 0xDA, 0xAA, 0xAE, 0xFB, 0xBA, 0x3A, 0xFD, 0xBF, 0xF1,

0xAC, 0x8D, 0x19, 0x6B, 0xC3, 0x13, 0xFC, 0x13, 0x09, 0x26, 0x81, 0x0C, 0x05, 0x50, 0x39, 0x50 }

Encrypted (wrapped) AES key:

C = 0x6CFD13B76436CD0DB70244FAE380CBA1

C[16] =

{ 0x6C, 0xFD, 0x13, 0xB7, 0x64, 0x36, 0xCD, 0x0D, 0xB7, 0x02, 0x44, 0xFA, 0xE3, 0x80, 0xCB, 0xA1 }

Authentication tag:

T = 0xc8bf18ac796b0b1d3a1256d3a91676c8

T[16] =

{ 0xC8, 0xBF, 0x18, 0xAC, 0x79, 0x6B, 0x0B, 0x1D, 0x3A, 0x12, 0x56, 0xD3, 0xA9, 0x16, 0x76, 0xC8 }

D.6.3 MAC1

Test vectors for MAC1 with SHA-256 (i.e., HMAC-SHA-256)

==========

Inputs: authentication key (K), message to be authenticated (M)

Output: Tag (T) of size 128 bits, i.e. 16 octets

Test Vector #1:

K = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b

key[20] =

{ 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B,

0x0B, 0x0B, 0x0B, 0x0B }

M = 0x4869205468657265

msg[8] =

{ 0x48, 0x69, 0x20, 0x54, 0x68, 0x65, 0x72, 0x65 }

T = 0xb0344c61d8db38535ca8afceaf0bf12b

tag[16] =

{ 0xB0, 0x34, 0x4C, 0x61, 0xD8, 0xDB, 0x38, 0x53, 0x5C, 0xA8, 0xAF, 0xCE, 0xAF, 0x0B, 0xF1, 0x2B }

Test Vector #2:

K = 0x4a656665

key[4] =

{ 0x4A, 0x65, 0x66, 0x65 }

M = 0x7768617420646f2079612077616e7420666f72206e6f7468696e673f

msg[28] =

{ 0x77, 0x68, 0x61, 0x74, 0x20, 0x64, 0x6F, 0x20, 0x79, 0x61, 0x20, 0x77, 0x61, 0x6E, 0x74, 0x20,

0x66, 0x6F, 0x72, 0x20, 0x6E, 0x6F, 0x74, 0x68, 0x69, 0x6E, 0x67, 0x3F }

T = 0x5bdcc146bf60754e6a042426089575c7

tag[16] =

{ 0x5B, 0xDC, 0xC1, 0x46, 0xBF, 0x60, 0x75, 0x4E, 0x6A, 0x04, 0x24, 0x26, 0x08, 0x95, 0x75, 0xC7 }

Test Vector #3:

K = 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

key[20] =

{ 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA }

M= 0xdddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd

msg[50] =

{ 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD,

0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD,

0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD,

0xDD, 0xDD }

T = 0x773ea91e36800e46854db8ebd09181a7

tag[16] =

{ 0x77, 0x3E, 0xA9, 0x1E, 0x36, 0x80, 0x0E, 0x46, 0x85, 0x4D, 0xB8, 0xEB, 0xD0, 0x91, 0x81, 0xA7 }

Test Vector #4:

K = 0x0102030405060708090a0b0c0d0e0f10111213141516171819

key[25] =

{ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10,

0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19 }

M= 0xcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcd

msg[50] =

{ 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD,

0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD,

0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD, 0xCD,

0xCD, 0xCD }

T = 0x82558a389a443c0ea4cc819899f2083a

tag[16] =

{ 0x82, 0x55, 0x8A, 0x38, 0x9A, 0x44, 0x3C, 0x0E, 0xA4, 0xCC, 0x81, 0x98, 0x99, 0xF2, 0x08, 0x3A }

Test Vector #5:

K = 0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c

key[20] =

{ 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C,

0x0C, 0x0C, 0x0C, 0x0C }

M = 0x546573742057697468205472756e636174696f6e

msg[20] =

{ 0x54, 0x65, 0x73, 0x74, 0x20, 0x57, 0x69, 0x74, 0x68, 0x20, 0x54, 0x72, 0x75, 0x6E, 0x63, 0x61,

0x74, 0x69, 0x6F, 0x6E }

T = 0xa3b6167473100ee06e0c796c2955552b

tag[16] =

{ 0xA3, 0xB6, 0x16, 0x74, 0x73, 0x10, 0x0E, 0xE0, 0x6E, 0x0C, 0x79, 0x6C, 0x29, 0x55, 0x55, 0x2B }

Test Vector #6:

K=
0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

key[131] =

{ 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA }

M=
0x54657374205573696e67204c6172676572205468616e20426c6f636b2d53697a65204b6579202d2048617
368204b6579204669727374

msg[54] =

{ 0x54, 0x65, 0x73, 0x74, 0x20, 0x55, 0x73, 0x69, 0x6E, 0x67, 0x20, 0x4C, 0x61, 0x72, 0x67, 0x65,

0x72, 0x20, 0x54, 0x68, 0x61, 0x6E, 0x20, 0x42, 0x6C, 0x6F, 0x63, 0x6B, 0x2D, 0x53, 0x69, 0x7A,

0x65, 0x20, 0x4B, 0x65, 0x79, 0x20, 0x2D, 0x20, 0x48, 0x61, 0x73, 0x68, 0x20, 0x4B, 0x65, 0x79,

0x20, 0x46, 0x69, 0x72, 0x73, 0x74 }

T = 0x60e431591ee0b67f0d8a26aacbf5b77f

tag[16] =

{ 0x60, 0xE4, 0x31, 0x59, 0x1E, 0xE0, 0xB6, 0x7F, 0x0D, 0x8A, 0x26, 0xAA, 0xCB, 0xF5, 0xB7, 0x7F }

Test Vector #7:

K=
0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

key[131] =

{ 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA }

M = 0x5468697320697320612074657374207573696e672061206c6172676572207468616e20626c6f636b2d73697a65206b657920616e642061206c6172676572207468616e20626c6f636b2d73697a6520646174612e2054686520 6b6579206e6565647320746f206265206861736865642062656666f72652062656696e6720757365642062792074686520484d414320616c676f726974686d2e

msg[152] =

{ 0x54, 0x68, 0x69, 0x73, 0x20, 0x69, 0x73, 0x20, 0x61, 0x20, 0x74, 0x65, 0x73, 0x74, 0x20, 0x75,

0x73, 0x69, 0x6E, 0x67, 0x20, 0x61, 0x20, 0x6C, 0x61, 0x72, 0x67, 0x65, 0x72, 0x20, 0x74, 0x68,

0x61, 0x6E, 0x20, 0x62, 0x6C, 0x6F, 0x63, 0x6B, 0x2D, 0x73, 0x69, 0x7A, 0x65, 0x20, 0x6B, 0x65,

0x79, 0x20, 0x61, 0x6E, 0x64, 0x20, 0x61, 0x20, 0x6C, 0x61, 0x72, 0x67, 0x65, 0x72, 0x20, 0x74,

0x68, 0x61, 0x6E, 0x20, 0x62, 0x6C, 0x6F, 0x63, 0x6B, 0x2D, 0x73, 0x69, 0x7A, 0x65, 0x20, 0x64,

0x61, 0x74, 0x61, 0x2E, 0x20, 0x54, 0x68, 0x65, 0x20, 0x6B, 0x65, 0x79, 0x20, 0x6E, 0x65, 0x65,

0x64, 0x73, 0x20, 0x74, 0x6F, 0x20, 0x62, 0x65, 0x20, 0x68, 0x61, 0x73, 0x68, 0x65, 0x64, 0x20,

0x62, 0x65, 0x66, 0x6F, 0x72, 0x65, 0x20, 0x62, 0x65, 0x69, 0x6E, 0x67, 0x20, 0x75, 0x73, 0x65,

0x64, 0x20, 0x62, 0x79, 0x20, 0x74, 0x68, 0x65, 0x20, 0x48, 0x4D, 0x41, 0x43, 0x20, 0x61, 0x6C,

0x67, 0x6F, 0x72, 0x69, 0x74, 0x68, 0x6D, 0x2E }

T = 0x9b09ffa71b942fcb27635fbcd5b0e944

tag[16] =

{ 0x9B, 0x09, 0xFF, 0xA7, 0x1B, 0x94, 0x2F, 0xCB, 0x27, 0x63, 0x5F, 0xBC, 0xD5, 0xB0, 0xE9, 0x44 }

D.6.4 KDF2

==========

Inputs: shared secret (ss), key derivation parameter (kdp), desired octet string length (dl)

Output: derived key of length dl octets

Test Vector #1:

---

ss = 0x96c05619d56c328ab95fe84b18264b08725b85e33fd34f08

ss[24] =

{ 0x96, 0xC0, 0x56, 0x19, 0xD5, 0x6C, 0x32, 0x8A, 0xB9, 0x5F, 0xE8, 0x4B, 0x18, 0x26, 0x4B, 0x08,

0x72, 0x5B, 0x85, 0xE3, 0x3F, 0xD3, 0x4F, 0x08 }

kdp = ""

dl = 16 octets

Test Vector #2:

---

ss = 0x96f600b73ad6ac5629577eced51743dd2c24c21b1ac83ee4

ss[24] =

{ 0x96, 0xF6, 0x00, 0xB7, 0x3A, 0xD6, 0xAC, 0x56, 0x29, 0x57, 0x7E, 0xCE, 0xD5, 0x17, 0x43, 0xDD,

0x2C, 0x24, 0xC2, 0x1B, 0x1A, 0xC8, 0x3E, 0xE4 }

kdp = ""

dl = 16 octets

Test Vector #3:

---

ss = 0x22518b10e70f2a3f243810ae3254139efbee04aa57c7af7d

ss[24] =

{ 0x22, 0x51, 0x8B, 0x10, 0xE7, 0x0F, 0x2A, 0x3F, 0x24, 0x38, 0x10, 0xAE, 0x32, 0x54, 0x13, 0x9E,

0xFB, 0xEE, 0x04, 0xAA, 0x57, 0xC7, 0xAF, 0x7D }

kdp = 0x75eef81aa3041e33b80971203d2c0c52

kdp[16] =

{ 0x75, 0xEE, 0xF8, 0x1A, 0xA3, 0x04, 0x1E, 0x33, 0xB8, 0x09, 0x71, 0x20, 0x3D, 0x2C, 0x0C, 0x52 }

dl = 128 octets

Test Vector #4:

---

ss = 0x7e335afa4b31d772c0635c7b0e06f26fcd781df947d2990a

ss[24] =

{ 0x7E, 0x33, 0x5A, 0xFA, 0x4B, 0x31, 0xD7, 0x72, 0xC0, 0x63, 0x5C, 0x7B, 0x0E, 0x06, 0xF2, 0x6F,

0xCD, 0x78, 0x1D, 0xF9, 0x47, 0xD2, 0x99, 0x0A }

kdp = 0xd65a4812733f8cdbcdfb4b2f4c191d87

kdp[16] =

{ 0xD6, 0x5A, 0x48, 0x12, 0x73, 0x3F, 0x8C, 0xDB, 0xCD, 0xFB, 0x4B, 0x2F, 0x4C, 0x19, 0x1D, 0x87 }

dl = 128 octets

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.[18, 19]

[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.[20]

[18] The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.
[19] IEEE publications are available from The Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).
[20] IETF documents (i.e., RFCs) are available for download at http://www.rfc-archive.org/.

[B14] ISO 3166-2:2013, Codes for the representation of names of countries and their subdivisions—Part 2: Country subdivision code.[21]

[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.

[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.[22]

[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.

[21] 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/).
[22] SAE publications are available from the Society of Automotive Engineers (http://www.sae.org/).