IEEE STANDARDS ASSOCIATION

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

Amendment 1

IEEE Vehicular Technology Society

Sponsored by the
Intelligent Transportation Systems Committee

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

IEEE Std 1609.2a™-2017
(Amendment to
IEEE Std 1609.2™-2016)

IEEE Std 1609.2a™-2017
(Amendment to
IEEE Std 1609.2-2016)

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

Amendment 1

Sponsor

Intelligent Transportation Systems Committee
of the
IEEE Vehicular Technology Society

Approved 28 September 2017

IEEE-SA Standards Board

Abstract: 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 are defined. Administrative functions necessary to support the core security functions are described.

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 978-1-5044-4270-1 STD22746
Print: ISBN 978-1-5044-4271-8 STDPD22746

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 Notices and Disclaimers Concerning IEEE Standards Documents." They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/IPR/disclaimers.html.

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

Photocopies

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/ 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 Chair*

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

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

Mohammad Battah
Aleksander Gogic
Gloria Gwynne
Ron Hochnadel
Carl Kain
John Kenney
Soyoung Kim
Jules Madey
Sean Maschue
Jim Misener
Randy Roebuck
Richard Roy
Kevin Smith
Jasja Tijink
Michaela Vanderveen
George Vlantis
Jason Wang
Ken Vaughn

When the IEEE-SA Standards Board approved this standard on 28 September 2017, it had the following membership:

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

Chuck Adams
Masayuki Ariyoshi
Ted Burse
Stephen Dukes
Doug Edwards
J. Travis Griffith
Michael Janezic
Thomas Koshy
Joseph L. Koepfinger*
Kevin Lu
Daleep Mohla
Damir Novosel
Ronald C. Petersen
Annette D. Reilly
Robby Robson
Dorothy Stanley
Adrian Stephens
Mehmet Ulema
Phil Wennblom
Howard Wolfman
Yu Yuan

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1609.2a-2017, IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages—Amendment 1.

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

- Since the publication of IEEE Std 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

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

Amendment 1

NOTE—The editing instructions contained in this amendment define how to merge the material contained therein into the existing base standard and its amendments to form the comprehensive standard.[1]

The editing instructions are shown in ***bold italic***. Four editing instructions are used: change, delete, insert, and replace. ***Change*** is used to make corrections in existing text or tables. The editing instruction specifies the location of the change and describes what is being changed by using ~~strikethrough~~ (to remove old material) and underscore (to add new material). ***Delete*** removes existing material. ***Insert*** adds new material without disturbing the existing material. Insertions may require renumbering. If so, renumbering instructions are given in the editing instruction. ***Replace*** is used to make changes in figures or equations by removing the existing figure or equation and replacing it with a new one. Editing instructions, change markings, and this NOTE will not be carried over into future editions because the changes will be incorporated into the base standard.

3. Definitions, abbreviations, and acronyms

3.1 Definitions

Delete the following definitions:

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

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

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

Insert the following definitions in alphanumeric order:

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.

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.

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

4. General description

4.2 Secure data service (SDS)

4.2.1 Secured protocol data units (SPDUs)

Change the last paragraph of 4.2.1 as follows:

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

4.2.2 Secure data service

4.2.2.3 Processing received SPDUs

4.2.2.3.1 Preprocessing

Change the last two paragraphs of 4.2.2.3.1 as follows:

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.

4.2.5 1609.2 security profile

Change the first paragraph of 4.2.5 as follows:

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.

4.3 Security services management entity (SSME)

4.3.1 General

Delete the final sentence of 4.3.1:

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

5. Cryptographic operations and validity

5.1 Certificate validity

5.1.1 Certificate contents

Change the third bulleted list in 5.1.1 as follows:

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 6.4.32 for a full specification):

— One or more PSIDs.

— For each PSID, the SSP Range, which identifies the SSPs associated with that PSID for which the CA is permitted to grant permissions.

— The permissible length(s) of the certificate chain from the certificate containing these issuance permissions to the certificate that signs the PDU. ~~This length is referred to as the permitted *depth*.~~ Certificate chain length is defined in 5.1.2.1.

— The end-entity type permissions, which indicate whether the ultimate end-entity certificate permits application operations, certificate request operations, or both.

Change the second paragraph following the third bulleted list in 5.1.1 as follows:

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* ~~*enrolment*~~ *certificate.*

Add a footnote to the second item in the fourth bulleted list in 5.1.1 as follows:

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*.[1] 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.

[1] 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].

5.1.2 Certificate chain

5.1.2.1 Certificate chain construction

Change the first five paragraphs of 5.1.2.1 and insert Figure 4a and Figure 4b as follows:

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 ~~certificate~~standard uses the terminology "B 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* ~~*the*~~ *a certificate chain* is defined as the number of certificates in the chain apart from the ~~certificate under consideration to~~topmost one, or equivalently as the ~~root, inclusive~~number of ~~both (so if the root were to sign a SPDU, the associated~~ intra-certificate ~~chain length would be 1). This is true even if the~~

~~local trust anchor used by a particular implementation is not the root but some intermediate CA.~~gaps in the chain. Figure 4a illustrates these two definitions. Figure 4b 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.

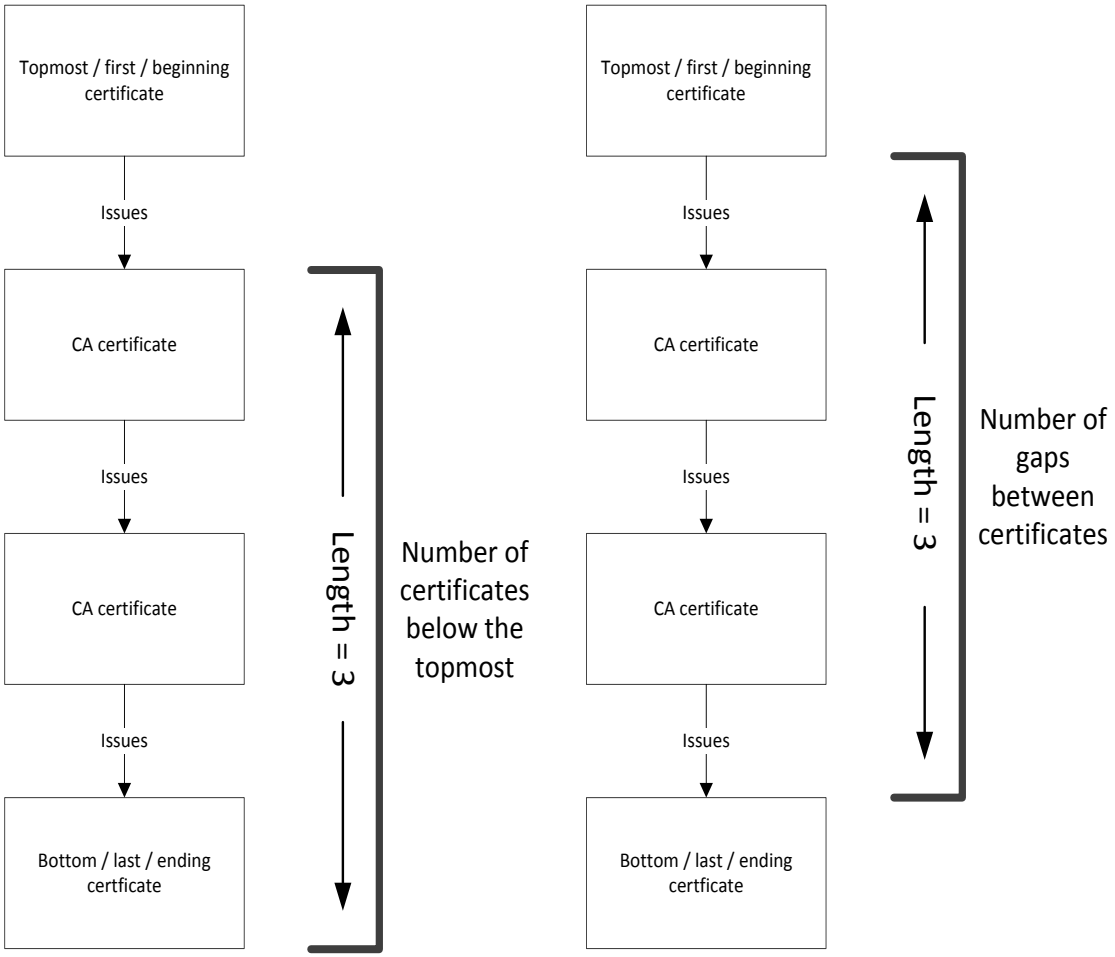

Figure 4a—A certificate chain of length 3

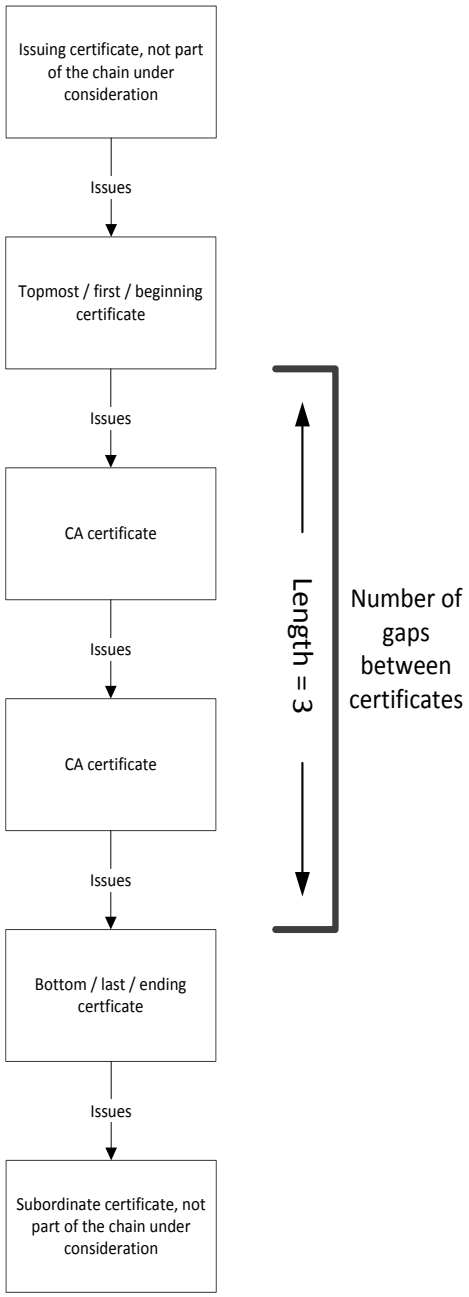

Figure 4b—A subchain of length 3 within a longer chain

5.1.2.2 Maximum supported certificate chain length

Change the text of 5.1.2.2 as follows:

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 ~~eight.~~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

Change the captions of Figure 5 and Figure 6 as follows:

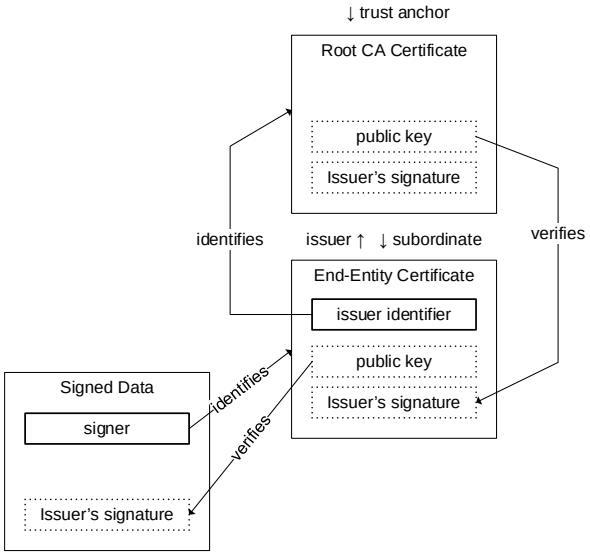

Figure 5—Cryptographic verification ~~with~~of a signed SPDU ~~and~~with a full certificate chain, using explicit certificates

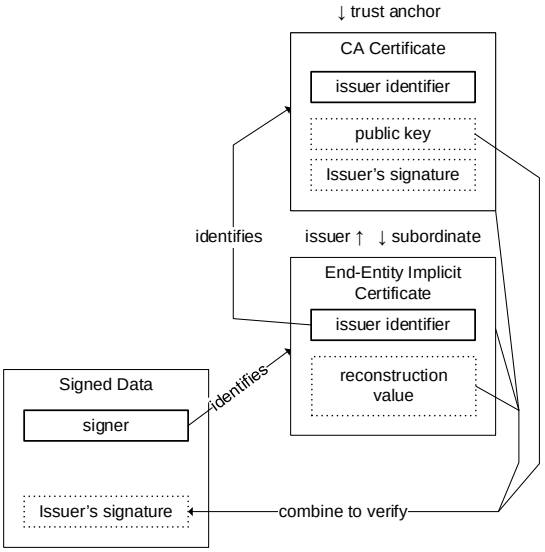

Figure 6—Cryptographic verification of a signed SPDU ~~and~~with a (non-full) certificate chain with implicit ~~certificates and a non-root CA as trust anchor~~end-entity certificate

5.1.2.4 Consistency of permissions within a certificate chain

Change the indicated text in 5.1.2.4 as follows:

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:

- **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:
  - ***[other bullet points omitted]***
  - If the subordinate entry is for certificate issuance ~~or request permissions~~, the permitted ~~depth~~length of the chain in the subordinate entry is consistent with the permitted ~~depth~~length of the chain in the issuing entry. Specifically, if ~~*minChainDepth*~~*minChainLength* and ~~*chainDepthRange*~~*chainLengthRange* in the subordinate certificate and issuing certificate~~s~~ have the values ~~mcd~~mcr_s, ~~cdr~~clr_s, ~~mcd~~mcr_i, ~~cdr~~clr_i, respectively, then ~~mcd~~$mcr_i \leq$ ~~mcd~~mcr_s+1 and (~~mcd~~mcr_i + ~~cdr~~clr_i) $\geq$ (~~mcd~~mcr_s + ~~cdr~~clr_s +1). (In the case where the subordinate certificate is an end-entity certificate, ~~mcd~~mcr_s and ~~cdr~~clr_s are set equal to zero (0) in these formulas.)

Insert 5.1.2.4a after 5.1.2.4:

5.1.2.4a 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

Change the first four paragraphs of 5.1.3.1 as follows:

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 ~~generation~~

~~time~~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 9.4.9.1 SSME-AddRevocationInfo.request and 9.4.9.6 SSME-AddRevocationInfo.confirm primitives. The SSME provides the revocation status of certificates via the 9.4.1.1 SSME-CertificateInfo.request and 9.4.1.6 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 ~~*R*~~, but there is one specific set that is identified as the one that will ~~indicate that~~ contain revocation information about C if it is revoked. ~~If *R* does not indicate that *C* is revoked, then *C* is not revoked.~~

~~Within the SSME~~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.

5.1.3.2 Determining which revocation information applies to a given certificate

Change the start of 5.1.3.2 as follows:

Revocation information applies to a given certificate if it:

— Indicates ~~the same~~ that certificate's *Certificate Revocation Authorizing CA* (CRACA) certificate, and

— Indicates ~~the same~~ that certificate's *CRL Series* value, and

— Is of the appropriate type (linkage-based or hash ID-based)

~~The~~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 ~~on~~one of the certificates in *C*'s ~~issuing~~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.

5.1.3.3 Identification of CRACA certificate

Change the text of 5.1.3.3 as follows:

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 6.4.3.

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 ~~it~~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 number of certificates in the chain that match the *cracaId* is anything other than 1,~~ the certificate is invalid.

5.1.3.6 Dubious certificates

Change the text of 5.1.3.6 as follows:

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`~~ 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

Change the text of 5.2.1 as follows, and insert the indicated figures:

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

The SDS supports two forms of signed SPDU: SPDUs signed with 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 with 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 10a illustrates the different validity conditions used to determine the validity of a signed SPDU that is signed with a certificate, and the input information used to check against those validity conditions. Figure 10b shows the information fields that go into creating a signed SPDU that is signed with a certificate.

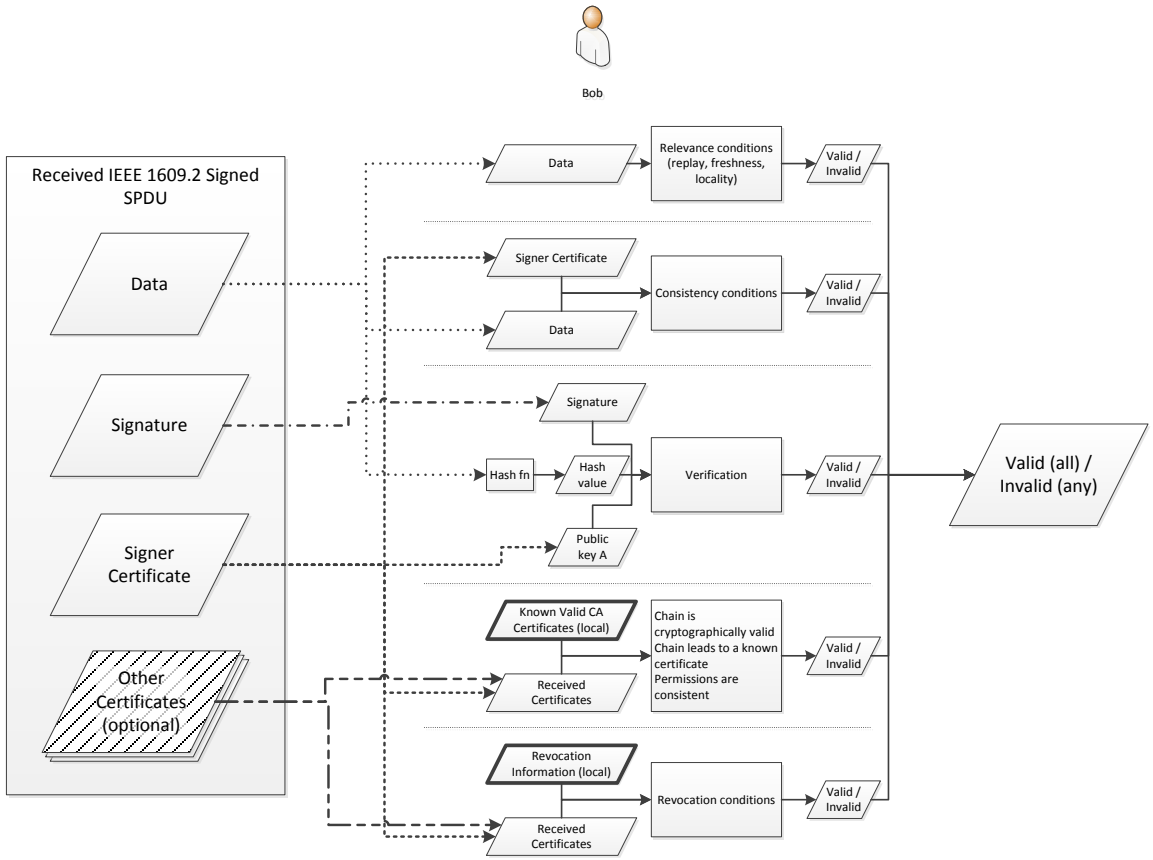

Figure 10a—Validity conditions for a signed SPDU

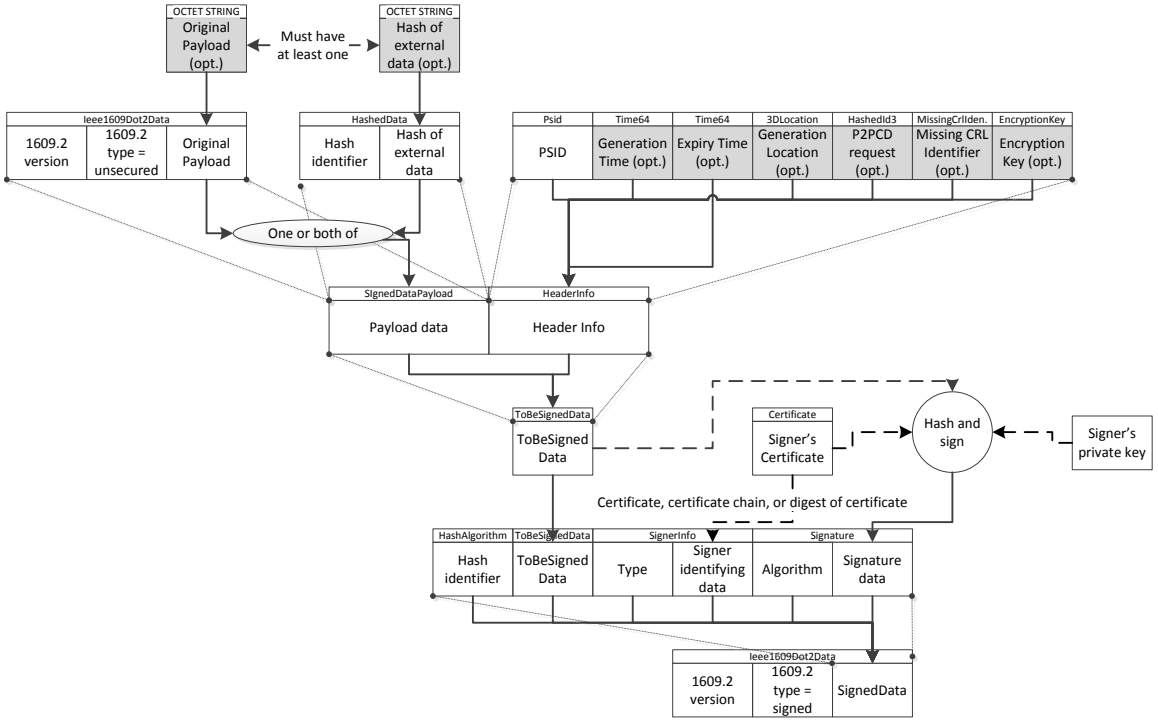

Figure 10b—Decomposition of a signed SPDU

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

Change the text of 5.2.3.2.1 as follows:

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 5.2.3.2.1a.

— The SPDU is internally consistent, as specified in 5.2.3.2.3.

— 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 5.2.3.2.2.

    — The PDU cryptographically verifies with the appropriate public keys. The cryptographic operations used for signing and verification are specified in 5.3.1. The encoding of data structures for input to those cryptographic operations is defined in Clause 6.

— ~~The PDU is internally consistent, as specified in 5.2.3.2.3.~~

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

If a signed SPDU does not meet the first three conditions and either the fourth (if it is self-signed) or the fifth (if it is signed with a certificate), it is invalid.

Insert 5.2.3.2.1a as follows:

5.2.3.2.1a 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.

5.2.3.2.2 Consistency between signed SPDU and signing certificate

Change the opening text of 5.2.3.2.2 as follows:

A signed SPDU that is signed with a certificate 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

Change the second set of bullet points in 5.2.3.2.2 as follows (only the changed bullet points are provided, all others are unchanged):

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

— […]

— 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:

  — […]

— The ~~stated generation time~~ 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.

5.2.3.3 SDEE-specific consistency conditions

5.2.3.3.1 General

Change 5.2.3.3.1 as follows:

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.

— (~~Optionally~~If signed with a certificate) The number of certificates in the full chain~~, from end entity~~ ending in the SPDU-signing certificate~~ to root certificate inclusive,~~ is less than some SDEE-specific limit (see Annex C). This condition can be verified by the SDS.

— (If signed with a certificate) 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 5.2.3.3.4a. This condition is not verified by the SDS and is intended to be verified by the receiving SDEE.

5.2.3.3.3 Consistency between SPDU payload and permissions: Service Specific Permissions

Change the opening sentence of 5.2.3.3.3 as follows:

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

Insert 5.2.3.3.4a and 5.2.3.3a after 5.2.3.3.4:

5.2.3.3.4a Consistency between SPDU payload and permissions: Relevance region

If a signed SPDU was signed with a certificate, the generation location consistency conditions specified in 5.2.3.2.2 can be used to determine that the 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.

5.2.3.3a Identified regions

In a valid signed SPDU signed with a certificate, 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

5.2.4.2 SDS-verified relevance conditions

5.2.4.2.1 General

Change the opening paragraph and bulleted list in 5.2.4.2.1 as follows:

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

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

The 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.

5.2.4.2.2 Generation time too far in the past

Change 5.2.4.2.2 as follows:

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

The 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 following algorithm is defined to determine whether the generation time of a signed SPDU is 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

Change 5.2.4.2.3 as follows:

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

The 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 following algorithm is defined to determine whether the generation time of a signed SPDU is 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

Change 5.2.4.2.4 as follows:

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

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

5.2.4.2.5 Generation location too distant

Change 5.2.4.2.5 as follows:

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

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

5.2.4.2.6 Replay

Change the contents of 5.2.4.2.6 as follows:

~~The SDS can be requested to detect whether a signed SPDU is an exact duplicate of one previously processed by the SDS for that SDEE. The replay detection service is provided by SSME.Sec-ReplayDetection.request, SSME.Sec-ReplayDetection.confirm. The replay detection service indicates that a signed SPDU is a replay if the entire encoded signed SPDU, including signature and other fields such as generation time inserted by the SDS, is identical to a recently received SPDU. The definition of "recently received" is SDEE-specific, but it is a logically consistent choice for this value to be the same as the value used to determine whether a SPDU has a generation time too far in the past (see 5.2.4.2.2), and the~~

~~interfaces defined in this standard enforce that the two values are the same. Other replay detection techniques, such as ones based on the payload only or on the same data encoded in different ways, are out of scope of this standard.~~

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

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 5.2.4.2.2), 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 subclause 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 subclause 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 following algorithm is defined 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 considered as invalid.

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

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

5.2.5 Supported critical information fields

Change the text of 5.2.5 as follows:

Critical information fields are any fields necessary to establish the validity of a signed SPDU. An implementation of WAVE Security Services that cannot ~~parse~~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 ~~parse~~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~~) or depth,~~), the definition in Clause 6 specifies the circumstances under which it is a critical information field, and a minimum size to be supported by any conformant implementation of WAVE Security Services. The Protocol Implementation Conformance Statement (PICS) provided in Annex A allows an implementation to state any size it supports beyond the minimum required for conformance.

5.3 Cryptographic operations

5.3.1 Signature algorithms

Change the second paragraph of 5.3.1 as follows:

~~Two~~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.

Change list entry c)2) in 5.3.1 as follows:

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.

Delete the second-last paragraph of 5.3.1:

~~The encoding of the signature is specified in 6.3.28.~~

5.3.2 Implicit certificates

Change 5.3.2 as follows:

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

a) In this standard, an implicit certificate is ~~encoded as~~ an ImplicitCertificate, as defined in 6.4.5, encoded with the Canonical Octet Encoding Rules (COER). All references to "the certificate CertU" in SEC 4 should be taken as referring to the encoded ImplicitCertificate except ~~when~~in the instance the implicit certificate is hashed to an integer ~~modulus~~ 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.

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 ~~within~~ 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.

Change the title and contents of 5.3.3 as follows:

5.3.3 Hash algorithms: SHA-256, SHA-384

The ~~only~~ hash ~~algorithm~~algorithms approved for use in this standard ~~is~~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".

5.3.5 Public key encryption algorithms: ECIES

Change the text of 5.3.5 as follows:

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. ~~Data structures and encoding rules for data objects associated with ECIES are specified in Clause 6 of this standard and include an indication of which curve is applicable.~~

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

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

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.

- ~~The public key *V* shall be freshly generated for each encryption operation.~~

  — 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. ~~This means that the elliptic curve points shall be converted to octet strings using LSB compressed representation.~~

  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*~~). The~~), where *c* is the COER encoding of the certificate~~ shall be put in,~~ canonicalized ~~form when hashing: see~~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.

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

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

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

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

5.3.8 Symmetric algorithms: AES-CCM

Insert the following text at the end of 5.3.8:

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

6. Data structures

6.1 Presentation and encoding

Change the third and fourth paragraphs of 6.1 as follows:

There are some data structures in this standard for which a “canonical encoding” is defined. This is the encoding to be used whenever the structures are to be encoded for processing by a cryptographic hash function. In general, these are structures that include the output of some cryptographic operation, for which the generator of the structure may choose either to include additional information to speed up receive-side processing, or to omit that additional information and reduce the transmitted packet size. ~~The structures for which a canonical encoding is defined are SignedData, ToBeSignedData, HeaderInfo, HashedId8EcdsaP256Signature, EcdsaP384Signature, CertificateBase, and~~

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

~~Clause 6 specifies and describes the data structures one at a time.~~ The complete IEEE 1609.2 ASN.1 modules are given in Annex B. In the event of a conflict between Annex B and this clause, this clause takes precedence.

Change the title of 6.2 as follows:

6.2 ~~Integer~~ Basic types

6.3 Secured protocol data units (SPDUs)

6.3.4 SignedData

Change the text of 6.3.4 as follows:

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

In this structure:

- `hashId` indicates the hash algorithm to be used to generate the hash of the message for signing and verification.
- `tbsData` contains the data that is hashed as input to the signature.
- `signer` determines the keying material and hash algorithm used to sign the data
- `signature` contains the digital signature itself, calculated as specified in 5.3.1~~, with:~~
  - 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

Change the ASN.1 code and first paragraph of 6.3.5 as follows:

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

This structure identifies a hash algorithm. The ~~only~~ value ~~currently supported is~~ `sha256`~~, indicating~~ 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.

6.3.9 HeaderInfo

Change the ASN.1 code, the indicated bullet points, and the "encoding considerations" paragraph in 6.3.9 as follows:

```
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,
}
```

— `p2pcdLearningRequest`, if present, is used by the SDS to request certificates for which it has seen identifiers but ~~do~~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 ~~have~~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 ~~communcation~~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, and6.3.36. 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 type EccP256CurvePoint or EccP384CurvePoint), then the elliptic curve point is encoded in compressed form, i.e., such that the choice indicated within the ~~EccP256CurvePoint~~Ecc*CurvePoint is `compressed-y-0` or `compressed-y-1`.

~~**Critical information fields**: This structure contains no critical information fields in the sense defined in 5.2.5~~

6.3.16 MissingCrlIdentifier

Change the text of 6.3.16 as follows:

```
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.

6.3.19 SymmetricEncryptionKey

Delete the following paragraph:

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.

6.3.21 SymmAlgorithm

Delete the following paragraph:

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.

6.3.22 BasePublicEncryptionKey

Delete the following paragraph:

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

6.3.23 EccP256CurvePoint

Change 6.3.23 as follows:

```
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 ~~LSB~~ least significant bit of y: if the ~~LSB~~ least significant bit of y is 0, type takes the value `compressed-y-0`, and if the ~~LSB~~ 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.

Insert new 6.3.23a:

6.3.23a 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.

6.3.24 SignerIdentifier

Change 6.3.24 as follows:

[…]

— If the choice indicated is `digest`:

  — The structure contains the HashedId8 of the relevant certificate~~, obtained as specified in the description of the HashedId8 structure~~. The 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.

6.3.25 HashedId3

Change 6.3.25 as follows:

```
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 ~~SHA-256~~ 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 ~~32 byte~~ 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.

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

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

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

6.3.26 HashedId8

Change the text of 6.3.26 as follows:

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

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

Example: Consider the SHA-256 hash of the empty string:

```
SHA-256("") =
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934c
```**a495991b7852b855**

The HashedId8 derived from this hash was highlighted above and corresponds to the following:

```
HashedId8 = a495991b7852b855.
```

6.3.27 HashedId10

Change the text of 6.3.27 as follows:

This data structure contains the truncated hash of another data structure. The HashedId10 for a given data structure is calculated by calculating the ~~SHA-256~~ hash of the encoded data structure and taking the low-order ten bytes of the hash output. The low-order ten bytes are the last ten bytes of the ~~32-byte~~ hash when represented in network byte order. If the data structure is subject to canonicalization it is canonicalized before hashing. 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.

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

Example: Consider the SHA-256 hash of the empty string:

```
SHA-256("") =
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b
```**934ca495991b7852b855**

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

6.3.28 Signature

Change the text of 6.3.28 as follows:

```
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.

6.3.29 EcdsaP256Signature

Change the text of 6.3.29 as follows:

```
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.[1]

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:

[1] 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.

NISTp256:

— p = FFFFFFFF000000010000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF

— n = FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

Brainpoolp256:

— p = A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377

— n = A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7

Insert 6.3.29a:

6.3.29a 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`.

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.[2]

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

[2] 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.

6.3.31 RecipientInfo

Change the text of 6.3.31 as follows:

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 ~~in~~was obtained from a certificate. ~~This field contains the HashedId8 of the certificate.~~ In this case, the parameter P1 to ECIES as defined in 5.3.5 is the hash of the certificate.

— `signedDataRecipInfo`: The data encryption key was encrypted using a public encryption key, where the encryption key was obtained as the public response encryption key from a SignedData. ~~In this case, this field contains the HashedId8 of the1609Dot2Data containing the SignedData containing the encryption key.~~ In this case, the parameter P1 to ECIES as defined in 5.3.5 is the SHA-256 hash of the Ieee1609Dot2Data containing the response encryption key.

— `rekRecipInfo`: The data encryption key was encrypted using a public ~~response encryption~~ key that was not obtained from a SignedData. ~~In this case, this field contains the HashedId8 of the resopnse encryption key.~~ In this case, the parameter P1 to ECIES as defined in 5.3.5 is the hash of the empty string.

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

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

Renumber the incorrectly numbered second subclause 6.3.1 to 6.3.32 and change its text as follows:

6.3.32 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.

Renumber the incorrectly numbered second subclause 6.3.2 to 6.3.33 and change its text as follows:

6.3.33 SymmRecipientInfo

— `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.

Renumber the incorrectly numbered second subclause 6.3.3 to 6.3.34 and change its text as follows:

6.3.34 PKRecipientInfo

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.

Renumber the incorrectly numbered second subclause 6.3.4 to 6.3.35 and change its text as follows:

6.3.35 EncryptedDataEncryptionKey

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

Renumber the incorrectly numbered second subclause 6.3.5 to 6.3.36.

Renumber the incorrectly numbered second subclause 6.3.6 to 6.3.37.

Renumber the incorrectly numbered second subclause 6.3.7 to 6.3.38.

Insert 6.3.38a:

6.3.38a 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 composed 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

6.4.3 CertificateBase

Change the "encoding considerations" section of 6.4.3 as follows:

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

— The encoding of `toBeSigned` uses the compressed form for all elliptic curve points: that is, those points ~~(which in this standard are all EccP256CurvePoints)~~ indicate a choice of `compressed-y-0` or `compressed-y-1`.

— The encoding of the signature, if present and if an ECDSA signature, takes the *r* value to be an EccP256CurvePoint or EccP384CurvePoint indicating the choice `x-only`.

Insert the following text at the end of 6.4.3:

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 an EccP256CurvePoint.

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

6.4.4 CertificateType

Insert the following text at the end of 6.4.4:

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.

6.4.7 IssuerIdentifier

Change 6.4.7 as follows:

```
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. ~~obtained as specified in the description of the HashedId8 structure~~.

— The hash algorithm to be used to generate the hash of the certificate for verification is SHA-256 (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 the ToBeSignedCertificate.

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

Change the text in 6.4.8 as follows:

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.

Change the encoding considerations text in 6.4.8 as follows:

Encoding considerations: The encoding of `toBeSigned` which is input to the hash uses the compressed form for all public keys and reconstruction values that are elliptic curve points: that is, those points ~~(which in this standard are all EccP256CurvePoints)~~ indicate a choice of `compressed-y-0` or `compressed-y-1`. The encoding of the issuing certificate uses the compressed form for all public key and reconstruction values and takes the *r* value of an ECDSA signature, which in this standard is an ~~EccP256CurvePoint~~ 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 SEC 4 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 ~~encrypted~~ 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 ~~encrypted~~ 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 ~~encrypted~~ signed SPDU as invalid. A compliant implementation shall support `certRequestPermissions` fields containing at least eight entries.

6.4.9 CertificateId

Change "but" to "and" in 6.4.9 as follows:

— `name` is used to identify the certificate holder in the case of non-anonymous certificates. The contents of this field are a matter of policy ~~but~~and should be human-readable.

6.4.17 GeographicRegion

Change the final bullet point in 6.4.17 as follows:

— If selected, `identifiedRegion` is a critical information field as defined in 5.2.5. An implementation that does not support the number of IdentifiedRegion in `identifiedRegion` shall reject the ~~encrypted~~signed SPDU as invalid. A compliant implementation shall support `identifiedRegion` fields containing at least eight entries.

6.4.24 CountryAndRegions

Change "verision" to "version" in 6.4.24 as follows:

— `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 ~~verision~~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

Change the text of 6.4.25 as follows:

— `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 ~~verision~~version of the 2010 FIPS codes as provided by the U.S. Census Bureau (see normative references in Clause 2). For other values of `country`, the meaning of `regionAndSubregions` is not defined in this version of this standard.

Critical information fields:

— If present, this is a critical information field as defined in 5.2.5. An implementation that does not ~~support the number of~~ recognize RegionAndSubregions ~~in the~~ 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

Insert the following text at the end of 6.4.26:

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.

6.4.28 PsidSsp

Delete the following sentences from the end of 6.4.28:

These permissions are PSID-specific. See Annex C for further discussion.

Insert the following text after the end of 6.4.28:

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

Change the ASN.1 and the following text in 6.4.29 as follows:

```
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.

Delete the following text from 6.4.29:

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.

Insert the following text at the end of 6.4.29:

Consistency with issuing certificate.

If a certificate has an `appPermissions` entry *A* for which the `ssp f`ield 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.

Insert 6.4.29a:

6.4.29a 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 *P* indicates `bitmapSspRange` and for every bit set to 1 in the `sspBitmask` in *P*, the bit in the identical position in the `sspValue` in *A* is set equal to the bit in that position in the `sspValue` in *P*.

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.

6.4.30 PsidGroupPermissions

Change the text in 6.4.30 as follows:

~~PsidGroupPermissions ::= SEQUENCE {~~

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

    SequenceOfPsidGroupPermissions ::= SEQUENCE OF PsidGroupPermissions
```

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

- `subjectPermissions` indicates PSIDs and SSP Ranges covered by this field.
- `minChainDepth` and `chainDepthRange` indicate how long the certificate chain from this certificate to the end entity certificate is permitted to be. The length of the certificate chain is measured from the certificate issued by this certificate to the end entity certificate in the case of `certIssuePermissions` and from the certificate requested by this certificate to the end entity certificate in the case of `certRequestPermissions`; a length of 0 therefore indicates that the certificate issued or requested is an end entity certificate. The length is permitted to be (a) greater than or equal to `minChainDepth` certificates and (b) less than or equal to `minChainDepth + chainDepthRange` certificates. The value −1 for `chainDepthRange` is a special case: if the value of `chainDepthRange` is −1 that indicates that the certificate chain may be any length equal to or greater than `minChainDepth`. See the examples below for further discussion.
- `eeType` takes one or more of the values `app` and `enrol` and indicates the type of certificates or requests that this instance of PsidGroupPermissions in the certificate is entitled to authorize. If this field indicates `app`, the chain ends may end in an authorization certificate, i.e., a certficate in which these permissions appear in an `appPermissions` field. If this field indicates `enrol`, the chain ends may end in an enrolment certificate, i.e., a certificate in which these permissions appear in a `certReqPermissions` permissions field), or both. Different instances of PsidGroupPermissions within a ToBeSignedCertificate may have different values for eeType.

Examples:

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

```
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 certficate 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 D.5.3 and D.5.4.

6.4.32 EndEntityType

Change 6.4.32 as follows:

```
  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 ~~ItsSspDepthRange~~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.

6.4.33 PsidSspRange

Change 6.4.33 as follows:

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

}

SequenceOfPsidSspRange ::= SEQUENCE OF PsidSspRange
```

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

— `psid` identifies the application area.

— `sspRange` identifies the SSPs associated with that PSID for which the holder may issue or request certificates. ~~If sspRange is omitted, the holder may only issue or request certificates for the deafult SSP for that psid~~. If sspRange is omitted, the holder may issue or request certificates for any SSP for that PSID.

6.4.34 SspRange

Change the contents of 6.4.34 as follows:

```
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.

— ~~If the choice indiated is opaque, the certificate holder may issue or request certificates with the listed SSPs for that PSID.~~

— ~~If the choice indicated is all, the holder may issue or request certificates for the any SSP for that PSID.~~

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

— ~~The issuing certificate SspRange is of type `opaque` and one of the entries in the range exactly matches the SSP in the subordinate certificate.~~

— ~~The issuing certificate SspRange is of type `all`.~~

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

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

— ~~The issuing certificate SspRange is of type `all`.~~

~~**Critical information fields**:~~

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

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

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`".

Insert 6.4.34a:

6.4.34a 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.

6.4.35 VerificationKeyIndicator

Insert the following text at the end of 6.4.35:

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.

6.4.40 PublicVerificationKey

Change the contents of 6.4.40 as follows:

```
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`.

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

7.3 Data structures

7.3.2 CrlContents

Change the contents of 7.3.2 as follows:

```
CrlContents ::= SEQUENCE  {
    version            Uint8 (1),
    crlSeries          CrlSeries,
    ~~cracaId~~crlCraca           HashedId8,
    issueDate          Time32,
    nextCrl            Time32,
    priorityInfo       CrlPriorityInfo,
    typeSpecific       CHOICE {
        fullHashCrl          ToBeSignedHashIdCrl,
        deltaHashCrl         ToBeSignedHashIdCrl,
        fullLinkedCrl        ToBeSignedLinkageValueCrl,
        deltaLinkedCrl       ToBeSignedLinkageValueCrl,
        ...
    }
}
```

— ~~cracaId~~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 ~~cracaId~~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.

— nextCrl contains the time when the next CRL with the same crlSeries and ~~cracaId~~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 (~~cracaId~~crlCraca, crlSeries) pair as specified in 5.1.3.6.

7.3.3 CrlPriorityInfo

Insert the following text at the end of 7.3.3

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

7.3.4 ToBeSignedHashIdCrl

Change the text in 7.3.4 as follows:

— crlSerial is a counter that increments by 1 every time a new full or delta CRL is issued for the indicated ~~cracaId~~crlCraca and crlSeries values.

7.3.5 HashBasedRevocationInfo

Change the text in 7.3.5 as follows:

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

7.3.6 ToBeSignedLinkageValueCrl

Change the text in 7.3.6 as follows:

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

— ~~`intervalWithinI`~~`indexWithinI` is a counter that is set to 0 for the first CRL issued for the indicated combination of ~~`cracaId`~~`crlCraca`, `crlSeries`, and `iRev`, and increments by 1 every time a new full or delta CRL is issued for the indicated ~~`cracaId`~~`crlCraca` and `crlSeries` values without changing `iRev`.

7.3.11 GroupCrlEntry

Change the text in 7.3.11 as follows:

— `iMax` indicates that for these certificates, revocation information need no longer be calculated once *iCert* > *iMax* as the holders are known to have no more valid certs for that (~~`cracaId`~~`crlCraca`, `crlSeries`) at that point.

7.4 CRL: 1609.2 Security envelope

7.4.2 Consistency criteria

Change the contents of 7.4.2 as follows:

A valid signed CRL meets the validity criteria of Clause 5. In addition, as discussed in 5.1.3.2 and illustrated in Figure 10, a valid signed CRL also meets one of the following conditions:

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

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

7.4.3 Service Specific Permissions and associated consistency criteria

7.4.3.3 CracaType

Change the contents of 7.4.3.3 as follows:

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

— If this takes the value `isCraca`, the ~~`cracaId`~~`crlCraca` field in the CrlContents structure is invalid unless it indicates the certificate that signs the CRL.

7.4.4 Security profile

7.4.4.1IEEE 1609.2 security profile identification

Insert a new row as the first content row in the table in 7.4.4.1:

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

Change the table in 7.4.4.2 as follows, by modifying three rows and inserting the indicated row immediately before "Signer Identifier Policy Type":

Field	Value	Notes
p2pcd_~~useInteractiveForm~~flavor	~~False~~ None	Full cert chain is attached
Signer Type Self	Prohibited	CRLs are signed with a certificate
Signer Identifier Policy Type	Simple	Signer type self is prohibited
Simple Signer Identifier Policy: Signer Identifier Cert Chain Length	~~-1~~1	1 is enough to get back to the CRACA

7.4.4.3 Receiving

Change one row and inset one row in the table in 7.4.4.3 as follows:

Field	Value	Notes
Maximum Full Certificate Chain Length	8	
Relevance: Replay	False	Replayed CRLs are not an attack
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

7.4.4.4 Security management

Change the contents of the table in 7.4.4.4 as follows:

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~~8	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~~

7.4.5 ASN.1

Change the PSID value in 7.4.5 as follows:

```
CrlPsid ::= Psid(~~135~~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

Change the text of 8.1 as follows:

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

P2PCD is initiated when a ~~device~~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 ~~device~~WAVE Security Services instance that received the trigger SPDU uses P2PCD learning requests to request peer ~~device~~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.

Insert the following paragraph at the end of 8.1:

The IEEE 1609.2 security profile (see Annex C) provides a means for SDEE specifiers 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

Change 8.2.1 as follows:

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~~s WAVE device~~ has not recently ~~sent~~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 P2PCD 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 14. In the figure, each box is a WAVE device or set of WAVE devices, with each device hosting the functional entities specified above. A breakdown of the information flows showing the roles played by each functional entity is given in the illustrative Figure 16.

a) The trigger SPDU sender, a WAVE device, sends a trigger SPDU which is received by the other WAVE devices including:
   1) The trigger SPDU receiver
   2) Other WAVE devices that will later play a responder role

3) Other WAVE devices that will later not play a responder role

b) One of the receivers of the trigger SPDU takes on the role of P2PCD requester and sends a P2PCD learning request. This is received by all the WAVE devices.

c) The original sender, and the other responders, all select a random backoff time and send responses once that backoff time has expired. 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. ~~After the third response the threshold number is reached and no more responses are sent~~.

Other patterns are possible, depending on how many possible requesters hear the original trigger SPDU, how many possible responders hear the request, and the order in which the responders respond.

8.2.2 Functional entities

Change the contents of 8.2.2 as follows:

- 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 SSME requests the SDS via the SSME Sec-SAP to include~~ 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.

8.2.3 Configuration parameters within SSME

Change the contents of 8.2.3 as follows:

P2PCD uses the following configuration parameters, which are managed by the SSME. These parameters may be SDEE-specific, or may be obtained from a system specification covering multiple SDEEs. They are configured by SSME-P2pcdConfiguration.request and SSME-P2pcdConfiguration.confirm. Recommended values are included in the discussion of these values in the send-side security profile in C.2.1.3.1.

— ~~`p2pcd_useInteractiveForm` (SDEE ID *s*): A Boolean indicating whether the interactive form of P2PCD is in use for the indicated SDEE.~~

— `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_useInteractiveForm` is true~~ `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

Insert a new subclause heading 8.2.4.1.1 within 8.2.4.1; move the contents of 8.2.4.1 to 8.2.4.1.1; change the text in 8.2.4.1.1 from the text originally in 8.2.4.1 as indicated below.

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:

Insert a new subclause, 8.2.4.1.2, as follows:

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-CertificateInfo.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

Insert new subclauses 8.2.4.2.1 and 8.2.4.2.2 within 8.2.4.2; move the text that was formerly in 8.2.4.2 to 8.2.4.2.1 and 8.2.4.2.2 as indicated below; change the text in 8.2.4.2.2 from the text originally in 8.2.4.2 as indicated below.

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 ~~interactive learning form~~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 ~~more~~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.

      i) ~~**Exception:** If the number of active responses is more than some implementation-specific amount, the SSME does not trigger response processing. The Protocol Implementation Conformance Statement (PICS) in A.2.3.3 allows suppliers to make a statement about the numbers supported by an implementation.~~

Insert 8.2.4.2.3 as follows:

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.

8.4 Data structures

8.4.1 P2PCD response message

8.4.1.1 ASN.1 definition

Change 8.4.1.1 as follows:

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)}

-- 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 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
```

9. Service primitives and functions

9.1 General comments and conventions

Change the first paragraph of 9.1 as follows:

Clause 9 specifies mechanisms for applying 1609.2 security processing to datagrams using primitives defined at Service Access Points (SAPs). The primitives defined at each SAP are summarized in Table 1 and specified in the indicated subclause. The details of the implementation of the primitives and their exchange protocols are not otherwise specified ~~but~~and are left as design decisions.

9.3.2 Sec-CryptomaterialHandle-GenerateKeyPair

9.3.2.1 Sec-CryptomaterialHandle-GenerateKeyPair.request

9.3.2.1.2 Semantics of the service primitive

Change the second row of the table in 9.3.2.1.2 as follows:

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

9.3.3 Sec-CryptomaterialHandle-StoreKeyPair

9.3.3.1 Sec-CryptomaterialHandle-StoreKeyPair.request

9.3.3.1.2 Semantics of the service primitive

Change the second row of the table in 9.3.3.1.2 as follows:

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

9.3.9 Sec-SignedData

9.3.9.1 Sec-SignedData.request

9.3.9.1.2 Semantics of the service primitive

Delete the parameter "Maximum Certificate Chain Length" from the list of parameters to the primitive in 9.3.9.1.2.

Delete the row with "Maximum Certificate Chain Length" from the table in 9.3.9.1.2.

Change the "Signer Identifier Cert Chain Length" and "Sign With Fast Verification" rows as follows:

Name	Type	Valid range	Description
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 ~~R in the EccP256CurvePoint~~`rSig` is set to `x-only`.

9.3.9.2 Sec-SignedData.confirm

9.3.9.2.3 When generated

Change list entry b)1)xx) in 9.3.9.2.3 as follows:

b) *Result Code*:

1) *Result Code* is set as follows if only one error occurred when signing:

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.

9.3.11.6 Sec-SecureDataPreprocessing.confirm

9.3.11.6.2 Semantics of the service primitive

Change the contents of 9.3.11.6.2 as follows:

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	~~Octet string~~A SSP of a type specified in 6.4.29	~~An octet string of length 0-32 octets~~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

Insert a new item (d) in the ordered list in 9.3.11.2.3 and renumber the items after it:

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.

9.3.12 Sec-SignedDataVerification

9.3.12.1 Sec-SignedDataVerification.request

9.3.12.1.2 Semantics of the service primitive

Change the contents of 9.3.12.1.2 as follows:

Sec-SignedDataVerification.request (
SDEE ID,
PSID,
~~*Content Type*,~~

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,
[…]
Maximum Full Certificate Chain Length (optional),
[…]
)

Name	Type	Valid range	Description
PSID	PSID	Any	The PSID derived from context (see 5.2.3.3.2).
~~*Content Type*~~	~~Enumerated~~	~~Signed~~ ~~Signed partial payload~~ ~~Signed external payload~~	~~The type of the 6.3.4 SignedData.~~
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)

9.3.12.2 Sec-SignedDataVerification.confirm

9.3.12.2.2 Semantics of the service primitive

Change the contents of 9.3.12.2.2 as follows:

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 […] SPDU-Certificate-Chain: Inconsistent chain permissions SPDU-Certificate-Chain: Inconsistent validity region SPDU-Crypto: Verification failure […]	The result of the validation operation
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

Change certain entries in the numbered list in 9.3.12.2.3 as follows:

i) "SPDU-Certificate-Chain: Chain ended at untrusted root" if the certificate chain can be constructed to a root, i.e., to a certificate with `issuer` indicating `self`, ~~but~~where that root is not trusted.

cc) "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.

Insert the following entry in the numbered list in 9.3.12.2.3 after the entry labelled o):

o.1) "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.

Insert the following text at the end of 9.3.12.2.3:

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.

9.4 SSME SAP

9.4.1 SSME-CertificateInfo

9.4.1.1 SSME-CertificateInfo.request

9.4.1.1.2 Semantics of the service primitive

Change the table in 9.4.1.1.2 as follows:

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

9.4.1.2 SSME-CertificateInfo.confirm

9.4.1.2.3 When generated

Change step a) 1) ii) in 9.4.1.2.3 by adding a footnote as follows:

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.[3]

9.4.6 SSME-AddHashIdBasedRevocation

9.4.6.1 SSME-AddHashIdBasedRevocation.request

9.4.6.1.2 Semantics of the service primitive

Change the contents of 9.4.6.1.2 as follows:

The parameters of the primitive are as follows:

SSME-AddHashIdBasedRevocation.request (
Identifiers,
~~*CracaId*~~*CrlCraca*,
CRL Series,
Expiry
)

[3] 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 9.4.1.1 SSME-CertificateInfo.request with that certificate as the *Identifier* parameter.

Name	Type	Valid range	Description
Identifiers	Array of HashedId10	As stated under Type	The HashedId10 values identifying the revoked certificates
~~*CracaId*~~*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

9.4.7 SSME-AddIndividualLinkageBasedRevocation

9.4.7.1 SSME-AddIndividualLinkageBasedRevocation.request

9.4.7.1.2 Semantics of the service primitive

Change the contents of 9.4.7.1.2 as follows:

The parameters of the primitive are as follows:

SSME-AddIndividualLinkageBasedRevocation.request (
~~*CracaId*~~*CrlCraca*,
CRL Series,
RevocationInfos
)

Name	Type	Valid range	Description
~~*CracaId*~~*CrlCraca*	HashedId8	An octet string of length 8	An identifier for the CRACA (see 5.1.3)

9.4.8 SSME-AddGroupLinkageBasedRevocation

9.4.8.1 SSME-AddGroupLinkageBasedRevocation.request

9.4.8.1.2 Semantics of the service primitive

Change the contents of 9.4.8.1.2 as follows:

The parameters of the primitive are as follows:

SSME-AddGroupLinkageBasedRevocation.request (
~~*CracaId*~~*CrlCraca*,
CRL Series,
RevocationInfos
)

Name	Type	Valid range	Description
~~*CracaId*~~*CrlCraca*	`HashedId8`	An octet string of length 8	An identifier for the CRACA (see 5.1.3)

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

Change the contents of 9.5.1.1.1 as follows:

This primitive allows any SDEE to determine whether received signed data is a ~~duplicate~~ 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

Change the contents of 9.5.1.1.2 as follows:

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 ~~SignedData~~ ToBeSignedData and signing certificate that ~~is~~ 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

Annex A

(informative)

Protocol Implementation Conformance Statement (PICS) proforma

A.2 PICS proforma—IEEE Std 1609.2[4]

A.2.3 Conformance statement

A.2.3.1 Security services

Change the table in Annex A.2.3.1 as indicated, renumbering items after inserted items.

Item	Security configuration (top-level)	Reference	Status	Support
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 other than ~~SHA-256~~	6.3.5	S1.2.2:O	□Yes □No
S.1.2.2.3.2.1	… … maximum number of certificates~~in the chain~~ included in the SignerIdentifier	6.3.25	S1.2.2.3.2 ~~8~~1:M > ~~8~~1:O	Enter number: ( )
S1.2.2.4.1.	… an ecdsa256Signature	6.3.31	S1.2.2.4:O6a	□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
S1.2.2.5.1.	Determine that the~~region is correct~~ generation location is consistent with the region in the certificate	5.2.3.2.2, 6.4.17	S1.2.2.5:~~O~~M	□Yes □No
S1.2.2.5.1.4.5.	List of supported IdentifiedRegions[5]	5.2.3.3a, 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: ~~O~~M	□Yes □No
S.1.2.2.5.3	Maximum supported length of the full chain (sending)	5.1.2.2	S1.2.2.5: 2:M >2:O	Enter number: ()
S1.3.2.1.1	Verify signed data using HashAlgorithm SHA-256	6.3.5	S1.3.2.1:O17a	□Yes □No

[4] 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.
[5] 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
S1.3.2.1.2	Verify signed data using HashAlgorithm SHA-384	6.3.5	S1.3.2.1:O17 a	□Yes □No
S1.3.2.1.3	Verify signed data using another HashAlgorithm	6.3.5	S1.3.2.1:O	□Yes □No
S.1.3.2.3.2.1	… … maximum number of certificates ~~in the chain~~ included in the SignerIdentifier	6.3.25	S1.3.2.3.2 ~~8~~1:M > ~~8~~1:O	Enter number: ( )
S1.3.2.4.2.	… an ecdsa256Signature	6.3.31	S1.3.2.4:O20 a	□Yes □No
S1.3.2.4.2.	… an ecdsa384Signature using Brainpool p384r1	6.3.31	S1.3.2.4:O20 a	□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
~~S1.3.2.5.1.7.~~	~~Maximum number of identifiedRegions supported~~	~~6.4.17, 6.4.22~~	~~S1.3.2.5.1.6: 8:M > 8:O~~	~~Enter number: ( )~~
S1.3.2.5.1.6.5.	List of supported IdentifiedRegions[6]	5.2.3.3a, 6.4.22	S1.3.2.5.1.6: M	Provide as additional information
S1.3.2.5.5.	Maximum supported length of the full chain (receiving)	5.1.2.2	S.1.2.2.5: 2: M <2: O	Enter number: ()
S1.3.3.2.5.	Containing pskRecipientInfo	6.3.33, 6.3.36	S1.3.3.2:O26	□Yes □No

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

A.2.3.3 Peer-to-peer certificate distribution (P2PCD) functionality

Change the table in A.2.3.3 as follows:

Item	Security configuration (top-level)	Reference	Status	Support
S3.	Support P2PCD	8	O	☐Yes ☐No
S3.1.	Number of supported SDEEs	8.2.6	S3.2: 1:O > 1:O	Enter number: ( )
S3.2.	**Support SSME and SDS operations for P2PCD in the requester role**	8.2.4.1	S3:O30	☐Yes ☐No
S3.2.1.	Under at least one condition, trigger request processing on receiving a trigger SPDU	8.2.4.1	S3.2:M	Enter description of at least one condition under which request processing is triggered ( )
S3.2.2.	Do not trigger request processing on receiving a trigger SPDU for which a request is already active	8.2.4.1	S3.2:M	☐Yes ☐No
S3.2.3.	Number of simultaneously active P2PCD learning requests	8.2.4.1, 8.2.6	S3.2: 1:O > 1:O	Enter number: ( )
S3.2.4.	When request processing is triggered, include a P2PCD learning request in the next SPDU for the trigger SDEE except in the following exception cases	8.2.4.1	S3.2: M	☐Yes ☐No
S3.2.4.1.	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	S3.2.4:O	☐Yes ☐No
S3.2.4.2.	Only include one P2PCD learning request no matter how many learning requests have been triggered	8.2.4.1	S3.2.4: M	☐Yes ☐No
S3.2.5.	Receive notifications from a P2PCDE that a P2PCD learning response has been received and use those to update the list of known certificates.	8.2.4.1	S3.2: M	☐Yes ☐No
S3.3.	**Support SSME and SDS operations for P2PCD in the responder role**	8.2.4.2	S3:O30	☐Yes ☐No
S3.3.1.	Trigger response processing on receiving a P2PCD learning request	8.2.4.2	S3.3:M	☐Yes ☐No
S3.3.2.	Number of simultaneously active P2PCD learning responses	8.2.4.1, 8.2.6	S3.3: 1:O > 1:O	Enter number: ( )
S3.3.3.	Do not trigger response processing if less than **`p2pcd_responseActiveTimeout`** has passed since last triggered	8.2.4.2	S3.3: M	☐Yes ☐No
S3.3.4.	Trigger sending response after random backoff time unless threshold number of responses have been observed	8.2.4.2	S3.3: M	☐Yes ☐No
S3.3.5.	Increment number of responses observed based on input from P2PCDE	8.2.4.2	S3.3: M	☐Yes ☐No
S3.4.	**Support P2PCDE operations for P2PCD**	8.2.4.2	S3:O30	☐Yes ☐No
S3.4.1.	Receive responses and provide to SSME	8.2.4.1, 8.2.4.2, 8.3.1	S3.4: M	☐Yes ☐No

~~S3.4.2.~~	~~Send responses when triggered by SSME~~	~~8.2.4.2, 8.3.1~~	~~S3.4: O~~	~~□Yes □No~~
~~S3.4.3.~~	~~Send responses over WSMP~~	~~8.2.4.2~~	~~S3.4.2: M~~	~~□Yes □No~~

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

Insert a new subclause B.0a as follows:

B.0a 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.1 1609.2 security services

B.1.1 1609.2 schema

Change B.1.1 as follows:

```
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


IMPORTS
  […]
EccP256CurvePoint,
~~  EciesP256EncryptedKey,~~
  EncryptedDataEncryptionKey,
  EncryptionKey,
  […]
  PublicVerificationKey,
  SequenceOfHashedId3,
  SequenceOfPsidSsp,
  […]
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)}

;

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,
}


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

PsidGroupPermissions ::= SEQUENCE  {
    appPermissionssubjectPermissions SubjectPermissions,
    minChainDepthLength     INTEGER DEFAULT 1,
    chainDepthLengthRange   INTEGER DEFAULT 0,
    eeType             EndEntityType DEFAULT {app}
}
```

B.1.2 1609.2 base types

Change B.1.2 as follows:

```
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

Uint64 ::= INTEGER (0..18446744073709551615) -- (hex) ff ff ff ff ff ff ff ff

SequenceOfUint3  ::= SEQUENCE OF Uint3
SequenceOfUint8  ::= SEQUENCE OF Uint8
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))
      }
  }


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

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

BitmapSsp ::= 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)),
}

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

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

B.2 Certificate revocation list (CRL)

B.2.1 Certificate revocation list: Base types

Change B.2.1 as follows:

```
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

IMPORTS
  […]
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 ::= SEQUENCE  {
    version            Uint8 (1),
    crlSeries          CrlSeries,
    cracaIdcrlCraca             HashedId8,
    issueDate          Time32,
    nextCrl            Time32,
    priorityInfo       CrlPriorityInfo,
    typeSpecific       CHOICE  {
      fullHashCrl          ToBeSignedHashIdCrl,
      deltaHashCrl         ToBeSignedHashIdCrl,
      fullLinkedCrl        ToBeSignedLinkageValueCrl,
      deltaLinkedCrl       ToBeSignedLinkageValueCrl,
      ...
    }
}


JMaxGroup ::= SEQUENCE  {
    jmax              Uint8,
    contents          SEQUENCE OF LAGroupSequenceOfLAGroup
}
```

B.2.2 CRL: Security envelope

Change B.2.2 as follows:

```
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

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(~~135~~256)
```

B.2.3 CRL: Service Specific Permissions (SSP)

Change B.2.3 as follows:

```
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)}

-- Minor version: 1

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

B.3 Peer-to-peer certificate distribution (P2PCD)

Change B.3 as follows:

```
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

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)}
;
```

Annex C

(informative)

Specifying the use of IEEE Std 1609.2™ by SDEEs

C.2 IEEE 1609.2 security profiles

C.2.1 Contents of security profile

Delete C.2.1.2:

~~**C.2.1.2 SDS**~~

C.2.1.3 IEEE 1609.2 security profile identification

Insert a new first content row to the table in C.2.1.3:

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.

C.2.1.3.1 Sending

Change the indicated entries in the table in C.2.1.3.1 as follows, and insert the indicated row ("Signer Type Self") immediately before "Signer Identifier Policy Type".

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

Name	Type	Recommended values	Description
Signer Type Self	Enumerated	"Required", "Permitted", "Prohibited"	Whether in the Ieee1609Dot2Data *d*, the field *d.content.signedData.signer* may take the value self.
Signer Identifier Policy Type	Enumerated	Simple Text	Describes the type of the Signer Identifier Policy. In the output signed SPDU, which is an Ieee1609-Dot2Data *d*, the Signer Identifier Policy indicates which option in the field *d.content.signedData.signer* is selected.

Name	Type	Recommended values	Description
			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.
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	~~Any~~ 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 9.3.9.1 Sec-Signed-Data.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 9.3.9.1 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

Name	Type	Recommended values	Description
			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_~~useInteractiveForm~~flavor	~~Boolean~~Enumerated	Inline Out of Band None	Whether to use the peer-to-peer certificate distribution defined in Clause 8.

Change certain of the bullet points in C.2.1.3.1 as follows. Insert the bullet point beginning "Signer Type Self" before the bullet point beginning "Signer Identifier Policy Type".

Guidance for SDEE specifiers:

— *Signer Type Self*, In general, *Signer Type Self* should be "Prohibited" and the other two fields can be omitted. If *Signer Type Self* is "Permitted" or "Required", a complete SDEE specification will indicate how the verification key is to be obtained by the verifier, for example from a particular field in the SPDU payload. If *Signer Type Self* is "Permitted", a complete SDEE specification will indicate the conditions under which the signer type may be *self*.

— *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.

— […]

— *p2pcd_~~useInteractiveForm~~flavor* and the interactive-form *p2pcd_** variables: in general it is recommended that SDEEs use ~~this~~ 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 ~~device~~ WAVE Security Services instance will send a request without regard to whether or not other ~~devices~~ instances are also requesting the same certificate.

Insert the following note after the bulleted list in C.2.1.3.1:

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

Change the "Use Preprocessing" and "Maximum Certificate Chain Length" table entries in C.2.1.3.2 as indicated:

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_useInteractiveForm* is True~~ *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.
Maximum Full Certificate Chain Length	Integer	Integer ≥ 2	The value set as *Maximum Full Certificate Chain Length* when invoking ~~Sec-SignedData.request and~~ Sec-SignedDataVerification.request.

Insert the following rows after "Generation Location Source" to the table in C.2.1.3.2:

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 5.2.3.3a, 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.

Change the bulleted list in C.2.1.3.2 as indicated:

— *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.

C.2.1.3.3 Security management

Change the indicated entries in the table in C.2.1.3.3 as follows:

Name	Type	Valid range	Description
Signing Key Algorithm	Enumerated	ecdsaNistP256withSha~~1~~256 ecdsaBrainpoolP256r1withSha~~1~~256	One of the valid signing algorithms identified in 5.3.1 and 6.4.40.
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.
Signature Algorithms in Chain or CRL	Sequence of Enumerated	One or more of: ecdsaNistP256withSha~~1~~256 ecdsaBrainpoolP256r1withSha~~1~~256 ecdsaBrainpoolP384r1withSha384	The signature algorithms that may be used in the certificate chain or to sign CRLs relevant to the application.

C.3 IEEE 1609.2 security profile proforma[7]

C.3.2 IEEE 1609.2 security profile proforma

C.3.2.2 Sending

Change the table in C.3.2.2 as follows:

[7] 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.

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 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_useInteractiveForm~~		
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

Change the table in C.3.2.3 as follows:

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

Change the table in C.3.2.4 as follows:

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.4.2 SSP syntax and semantics

Insert the following text at the end of C.4.2:

SDEE specifiers may choose to use SSPs that are opaque or in the form 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.

Insert C.7 as follows:

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.5 Example data structures

Insert D.5.3 and D.5.4 as follows:

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
```

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

Insert D.6 as follows:

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 = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b

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 =
0x5465737420557369 6e67204c6172676572205468616e20426c6f636b2d53697a65204b6579202d2048617368204b6579204669727374

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 =
0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
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 =
0x5468697320697320612074657374207573696e672061206c6172676572207468616e20626c6f636b2d736
97a65206b657920616e642061206c6172676572207468616e20626c6f636b2d73697a6520646174612e2054
6865206b6579206e6565647320746f20626520686173686564206265666f7265206265696e6720757365642
062792074686520484d414320616c676f726974686d2e

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
}

Consensus

WE BUILD IT.

Connect with us on:

Facebook: https://www.facebook.com/ieeesa

Twitter: @ieeesa

LinkedIn: http://www.linkedin.com/groups/IEEESA-Official-IEEE-Standards-Association-1791118

IEEE-SA Standards Insight blog: http://standardsinsight.com

YouTube: IEEE-SA Channel

IEEE
standards.ieee.org
Phone: +1 732 981 0060 Fax: +1 732 562 1571