



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

Intelligent Transportation Systems Committee

Sponsored by the
IEEE Vehicular Technology Society

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

6 July 2006

IEEE Std 1609.2™-2006

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

Sponsor

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

Approved 8 June 2006

IEEE-SA Standards Board

Abstract: Secure message formats, and the processing of those secure messages, within the DSRC/WAVE system are defined. The standard covers methods for securing WAVE management messages and application messages, with the exception of vehicle-originating safety messages. It also describes administrative functions necessary to support the core security functions

Keywords: authentication, Dedicated Short Range Communications (DSRC), encryption, vehicular communications, Wireless Access in Vehicular Environments (WAVE)

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

Copyright © 2006 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 6 July 2006. Printed in the United States of America.

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

Print: ISBN 0-7381-5008-8 SH95558
PDF: ISBN 0-7381-5009-6 SS95558

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

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

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS.**”

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. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, 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 publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received 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 interpretation requests except in those cases where the matter has previously received formal consideration. 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, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

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

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

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

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

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

With this in mind, at the time that IEEE P1609 was established to develop the standards for the DSRC wireless network stack, the IEEE also established IEEE P1556TM (later renumbered as IEEE 1609.2) to develop standards for the security techniques that will be used to protect the services that use this network stack. These applications face unique constraints. Many of them, particularly safety applications, are time-critical: the processing and bandwidth overhead due to security must be kept to a minimum, to improve responsiveness and decrease the likelihood of packet loss. For many applications, the potential audience consists of all vehicles on the road in North America; therefore, the mechanism used to authenticate messages must be as flexible and scalable as possible, and must accommodate the smooth removal of compromised units from the system. Additionally, as mentioned above, the privacy of privately owned and operated vehicles must be respected as far as technically and administratively feasible.

This document specifies a range of security services for use in the WAVE environment. Mechanisms are provided to authenticate WAVE management messages, to authenticate messages that do not require anonymity, and to encrypt messages to a known recipient. Mechanisms to provide anonymity, particularly anonymous broadcast, will be provided in a separate document.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/interp/index.html>.

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 with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and nondiscriminatory, reasonable terms and conditions to applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates, terms, and conditions of the license agreements offered by patent holders or patent applicants. Further information may be obtained from the IEEE Standards Department.

Publication of this trial-use standard for comment and criticism has been approved by the Institute of Electrical and Electronics Engineers. Trial-use standards are effective for 24 months from the date of publication. Comments for revision will be accepted for 18 months after publication. Suggestions for revision should be directed to the Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, and should be received no later than 7 January 2007. It is expected that following the 24-month period, this trial-use standard, revised as necessary, shall be submitted to the IEEE-SA Standards Board for approval as a full-use standard.

Participants

The active participants in the IEEE P1609.2 (1556) working group at the time this trial-use standard was developed and balloted were as follows:

Thomas M. Kurihara, *Chair*

Roger J. O'Connor, *Working Group Vice Technical Chair*

Douglas M. Kavner, *Security SWG Chair*

Eric Rescorla, *Primary Security Consultant*

William Whyte, *P1609.2 (1556) Technical Editor*

Scott Andrews
Lee R. Armstrong
Daniel V. Bailey
Jim Bauer
Art Carter
Broady Cash
Ronald K. Char
Emily Clark
J. Kenneth Cook
Khaled Dessouky
Eskafi Farokh
Wayne Fisher
Ramez Gerges
Susan Graham
Gloria Gwynne
Chris Hedges
Russell D. Housley

Mary Ann Ingram
Daniel Jiang
Carl Kain
Pankaj R. Karnik
Douglas M. Kavner
David Kelley
Hariharan Krishnan
Jeremy A. Landt
Jason Liu
Justin McNew
John T. Moring
Sean O'Hara
Peter Oomen
Sam Oyama
Joon Gou Park
Gordon Peredo
Frank Perry

Mohan Pundari
Ed Ring
Tom Schaffnit
Dick Schnacke
Douglas Siesel
Robert T. Soranno
Steve Spenler
Bill Spurgeon
Steve Tengler
Jim Tomcik
Roger Tong
Glenn Turnock
Bryan Wells
Filip Weytjens
Doug Whiting
Chris Wilson
Jijun Yin
Jeffery Zhu

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

Toru Aihara
Scott Andrews
Lee R. Armstrong
John R. Barr
Alexei Beliaev
Juan C. Carreon
Yi-ming Chen
Danila Chernetsov
Elizabeth Chesnutt
Kai Moon Chow
Keith Chow
J. Kenneth Cook
Tommy P. Cooper
Thomas J. Dineen
Randall L. Dotson
Marc Emmelmann
Avraham Freedman
Ignacio Marin Garcia
Nikhil Goel
Sergiu R. Goma
Randall C. Groves
Pradeep Gupta
Gloria G. Gwynne
Gary A. Heuston
Werner Hoelzl

Russell D. Housley
Raj Jain
Oh Jongtaek
Avinash Joshi
Pankaj R. Karnik
Piotr Karocki
Douglas M. Kavner
Stuart J. Kerry
Patrick W. Kinney
Jim Kulchisky
Thomas M. Kurihara
Jeremy A. Landt
Jun Liu
William Lumpkins
G. L. Luri
Julius M. Madey
Gary L. Michel
William J. Mitchell
Apurva N. Mody
Yasser L. Morgan
John T. Moring
Ross A. Morris
Andrew F. Myles
Michael S. Newman
Richard H. Noens

Satoshi Obara
Roger J. O'Connor
Chris L. Osterloh
Satoshi Oyama
Subburajan Ponnuswamy
Henry S. Ptasinski
Vikram Punj
Robert A. Robinson
Frank H. Rocchio
Randal D. Roebuck
Michael Scholles
Stephen C. Schwarm
Rich Seifert
John W. Sheppard
Robert T. Soranno
Luca Spotorno
Thomas E. Starai
Mark A. Tillinghast
Scott A. Valcourt
Christopher G. Ware
William Whyte
Eric V. Woods
Paul R Work
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 8 June 2006, it had the following membership:

Steve M. Mills, *Chair*
Richard H. Hulett, *Vice Chair*
Don Wright, *Past Chair*
Judith Gorman, *Secretary*

Mark D. Bowman
Dennis B. Brophy
William R. Goldbach
Arnold M. Greenspan
Robert M. Grow
Joanna N. Guenin
Julian Forster*
Mark S. Halpin
Kenneth S. Hanus

William B. Hopf
Joseph L. Koepfinger*
David J. Law
Daleep C. Mohla
T. W. Olsen
Glenn Parsons
Ronald C. Petersen
Tom A. Prevost

Greg Ratta
Robby Robson
Anne-Marie Sahazizian
Virginia C. Sulzberger
Malcolm V. Thaden
Richard L. Townsend
Walter Weigel
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan H. Cookson, *NIST Representative*

Michelle Turner
IEEE Standards Program Manager, Document Development

Matthew Ceglia
IEEE Standards Program Manager, Technical Program Development

Contents

1. Overview	1
1.1 Introduction	1
1.2 Scope	1
1.3 Purpose	2
1.4 Document organization.....	2
1.5 Document conventions	3
2. Normative references.....	3
3. Definitions, abbreviations, and acronyms	4
3.1 Definitions	4
3.2 Abbreviations and acronyms	9
3.3 Terminology, applications, implementations, and the security manager.....	10
4. Presentation language.....	11
4.1 General	11
4.2 Notation conventions.....	11
4.3 Basic block size	12
4.4 Numbers	12
4.5 Fixed-length vectors	12
4.6 Variable-length vectors.....	13
4.7 The opaque and opaqueExtLength type.....	13
4.8 Enumerated type.....	14
4.9 Constructed types	15
4.10 The case statement.....	15
4.11 The extern statement.....	16
4.12 Flags	16
5. Secured Messages.....	17
5.1 General	17

5.2 SecuredMessage type	17
5.3 SignedMessage, ToBeSignedMessage, and MessageFlags types.....	18
5.4 SignedWSM and ToBeSignedWSM types	19
5.5 PublicKey, PKAlgorithm, and SymmAlgorithm types.....	20
5.6 ECPublicKey type	20
5.7 CertID8 and CertID10 type	21
5.8 The ApplicationID and FullySpecifiedAppID types	21
5.9 Time64 and Time32 types	22
5.10 SignerInfo type	22
5.11 Signature type.....	23
5.12 ECDSASignature type.....	23
5.13 EncryptedMessage, EncryptedContentInfo, and RecipientInfo types	23
5.14 ECIESNISTp256EncryptedKey and AESCCMCiphertext types.....	24
5.15 WAVECertificate, ToBeSignedWAVECertificate, CertSpecificData, SubjectType, and CRLSeries types	25
5.16 WAVECRL, ToBeSignedCRL, CRLType, and IDAndDate types	27
5.17 WAVECertificateRequest and WAVECertificateResponse types.....	28
5.18 GeographicRegion and RegionType types	29
5.19 The 2DLocation and 3DLocationAndConfidence types.....	30
5.20 Certificate Scopes.....	31
6. Other secured message formats	34
7. Secure message processing.....	35
7.1 Required information for security services.....	35
7.2 Caches and stores	35
7.3 Signed messages.....	37
7.4 Processing Encrypted Messages.....	43
7.5 Processing Signed and Encrypted Messages	45
8. Specific uses of secured messages.....	45

8.1 Secured WSAs.....	45
8.2 Secured WSMs	50
8.3 Security Manager.....	53
8.4 Certificate requests	55
8.5 Fragmented messages	57
Annex A (normative) Protocol Implementation Conformance Statement (PICS) proforma.....	59
Annex B (normative) Summary of message formats.....	76
Annex C (informative) Examples of message structures.....	83
Annex D (informative) General description	89
Annex E (informative) Additional security considerations	95
Annex F (informative) Threat model.....	98
Annex G (informative) Bandwidth considerations and opportunities for optimization.....	101
Annex H (informative) Copyright statement for Clause 4.....	103
Annex I (informative) Bibliography	104

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

1. Overview

1.1 Introduction

Wireless Access in Vehicular Environments (WAVE) is a radio communications system intended to provide interoperable wireless networking services for transportation. These services include those recognized for Dedicated Short-Range Communications (DSRC) by the U.S. National Intelligent Transportation Systems (ITS) Architecture (NITSA) [B21]¹ and many others not specifically identified in the architecture. The system enables vehicle-to-vehicle (V2V) and vehicle-to-roadside or vehicle-to-infrastructure (V2I) communications, generally over line-of-sight distances of less than 1000 m, where the vehicles may be moving at speeds up to 140 km/h.

The Physical Layer (PHY) and Medium Access Control (MAC) use elements of the IEEE 802.11TM PHY and MAC and were under development at the time this standard was issued. Channelization and the upper layers of the network stack are defined in IEEE P1609.4TM [B4] and IEEE P1609.3TM, respectively. IEEE P1609.1TM [B3] defines an application, the Resource Manager, that uses the network stack for communications. This document, IEEE Std 1609.2, specifies security services for the WAVE networking stack and for applications that are intended to run over that stack. Services include encryption using another party's public key, and non-anonymous authentication.

1.2 Scope

The scope of this standard is to define secure message formats, and the processing of those secure messages, within the DSRC/WAVE system. The standard covers methods for securing WAVE

¹ Numbers in brackets correspond to the bibliography in Annex I.

management messages and application messages, with the exception of vehicle-originating safety messages. It also describes administrative functions necessary to support the core security functions.

1.3 Purpose

The safety-critical nature of many DSRC/WAVE applications makes it vital that services be specified that can be used to protect messages from attacks such as eavesdropping, spoofing, alteration, and replay. Additionally, the fact that the wireless technology will be deployed in personal vehicles, whose owners have a right to privacy, means that in as much as possible the security services must be designed to respect that right and not leak personal, identifying, or linkable information to unauthorized parties. This standard describes security services for WAVE management messages and application messages, with the exception of vehicle-originating safety messages, to meet these requirements. It is anticipated that vehicle-originating safety messages will be added in an amendment to this standard.

1.4 Document organization

The document contains both normative and informative text, and is organized as follows.

Clause 1 reviews the scope and purpose of this standard and introduces the main entities in the system.

Clause 2 contains the normative references. These are documents that shall be referred to in order to produce a compliant implementation of this standard.

Clause 3 provides a list of definitions and commonly used abbreviations.

Clause 4 describes the presentation language that is used in later clauses to define secured message formats.

Clause 5 uses the presentation language of Clause 4 to describe the secure message formats for time or bandwidth constrained applications. These include signed messages, encrypted messages, and digital certificates.

Clause 6 describes other mechanisms that may be used to secure messages.

Clause 7 describes the steps to be taken on sending or receiving one of the secured messages described in Clause 5.

Clause 8 describes how the secure messages are to be used in specific contexts.

Annex A is a Protocol Implementation Conformance Statement (PICS) proforma, to be completed by implementers of this standard to indicate with what parts of the standard their implementation is conformant.

Annex B collects all the secured message formats defined in the document in a single location for convenience.

Annex C gives examples of the structure of some encoded messages for the convenience of implementers.

Annex D provides an informative overview of the entire WAVE system.

Annex E describes additional security considerations. These are typically infrastructure considerations that are not necessary for interoperation of two devices conforming to this standard.

Annex F describes how specific applications might make different uses of the security services described in this document.

Annex G describes how to use the techniques described in this standard in a way that minimizes the size (and so maximizes the probability of successful transmission) of time-critical messages.

Annex H is a copyright statement covering the text from Clause 5 that was excerpted from IETF RFC 2246 [B11].

Annex I contains the bibliography. The documents referenced here may be used to gain a fuller understanding of this standard, but conformance to this standard does not depend on conformance to any of these documents.

1.5 Document conventions

Unless otherwise stated, conventions follow those in IEEE Std 802.11 [B6], including conventions for the ordering of information within data streams.

Numbers are decimal unless otherwise noted. Numbers preceded by 0x are to be read as hexadecimal, so that 0xFF is equivalent to “FF hexadecimal”.

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

2. Normative references

The following referenced documents are indispensable for the application of this standard. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

Federal Information Processing Standard (FIPS) 180-1, Secure Hash Standard.²

Federal Information Processing Standard (FIPS) 186-2, Digital Signature Standard.

IEEE P1609.3TM (Draft 19, March 2006), Draft Standard for Wireless Access in Vehicular Environments (WAVE) Networking Services.³

IEEE Std 1363TM-2000, IEEE Standard Specifications for Public Key Cryptography.^{4, 5}

IEEE Std 1363aTM-2004, IEEE Standard Specifications for Public Key Cryptography: Additional Techniques.

IETF Request for Comments: 768, User Datagram Protocol.⁷

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

³ This IEEE standards project was not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE (<http://standards.ieee.org>). Upon approved by the IEEE-SA Standards Board, IEEE P1609.3 should be superseded by IEEE Std 1609.3-200X (where 200X is the year of approval).

⁴ IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA (<http://standards.ieee.org/>).

⁵ The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

⁷ Internet RFCs are retrievable by FTP at [ds.internic.net/rfc/rfcnnnn.txt](ftp://ds.internic.net/rfc/rfcnnnn.txt) (where nnnn is a standard's publication number such as 768), or call InterNIC at 1-800-444-4345 for information about receiving copies through the mail.

IETF Request for Comments: 3629, UTF-8, A Transformation Format of ISO 10646.

NIMA Technical Report TR8350.2, “Department of Defense World Geodetic System 1984, Its Definition and Relationships With Local Geodetic Systems.”⁸

NIST Special Publication SP 800-38C, Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality.⁹

3. Definitions, abbreviations, and acronyms

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B1] should be referenced for terms not defined in this clause.

- 3.1.1 advanced encryption standard (AES):** A symmetric block cipher adopted by the U.S. government as a standard and used to encrypt and decrypt electronic data.
- 3.1.2 airlink:** A radio-frequency communication interface, from the physical layer of one device to that of a remote device, such as the interface defined by WAVE.
- 3.1.3 application:** In the context of IEEE Std 1609.2, a receiving application aggregates messages sent to one or more endpoints. A sending application sends messages to one or more endpoints.
- 3.1.4 application context mark (ACM):** A code that identifies a specific instance of an application.
- 3.1.5 application class identifier (ACID):** A code that identifies a class of application.
- 3.1.6 ApplicationID:** A structure that contains an ACID and an ACM.
- 3.1.7 associated ApplicationID:** If a certificate or signed message uses the `from_issuer` value in its ApplicationID, the associated ApplicationID is the first ApplicationID other than `from_issuer` to appear in its certificate chain. The associated ApplicationID is used as if it appeared in the original certificate or signed message.
- 3.1.8 asymmetric cryptographic algorithm:** A cryptographic algorithm that uses two related keys— a public key and a private key. The two keys have the property that given the public key it is computationally infeasible to derive the private key.
- 3.1.9 block cipher:** A symmetric encryption algorithm that processes data in blocks, typically of 8 or 16 octets.
- 3.1.10 broadcast application:** An application that transmits information without targeting a specific recipient and without expecting a response. *Contrast:* **transactional application.**
- 3.1.11 certificate authority (CA) certificate:** A certificate that is used to sign other certificates. *Contrast:* message signing certificate.

⁸ This technical report is available from http://earth-info.nga.mil/GandG/tr8350/tr8350_2.html.

⁹ This special publication is available from <http://csrc.nist.gov/publications/nistpubs/>.

- 3.1.12 cache:** A collection of data objects such that each object stays in the cache for a limited period of time.
- 3.1.13 certificate:** *See* **digital certificate**.
- 3.1.14 certificate authority (CA):** An entity that issues certificates, and is accepted by other entities to have in place valid procedures to ensure that, if a certificate identifies an entity, it is only issued to that entity, and if a certificate grants permissions, it is only issued to entities who have a right to be granted those permissions.
- 3.1.15 certificate chain:** A collection of certificates that can be put into an ordered list such that each certificate in the list (except the first) was issued by the owner of the certificate before it.
- 3.1.16 certificate signing request (CSR):** A request by an entity to a CA to be issued with a certificate.
- 3.1.17 certificate signing request signing certificate (CSRSC):** A certificate used to sign a certificate signing request (CSR).
- 3.1.18 certificate revocation list (CRL):** A mechanism for the revocation of certificates. *See:* **revocation**.
- 3.1.19 certificate subject:** The entity identified in a certificate; the entity that knows the private key corresponding to the public key in the certificate.
- 3.1.20 cipher block chaining (CBC):** A mode of operation for a block cipher that uses the previous block of ciphertext to randomize the current block of plaintext, masking plaintext patterns.
- 3.1.21 cipher block chaining message integrity check (CBC-MIC):** An application of the CBC mode of operation that produces a message integrity check rather than ciphertext.
- 3.1.22 control channel (CCH):** A radio channel used for exchange of management data and WAVE short messages.
- 3.1.23 counter mode:** A mode of operation of a block cipher where the output of encrypting an incrementing counter is used as a keystream.
- 3.1.24 cryptographically secure hash function:** A hash function is a transformation that converts a variable length input to a fixed length string called the hash value. A cryptographically secure hash function maps an arbitrary-length input into a fixed-length output (the hash value) such that (a) it is computationally infeasible to find an input that maps to a specific hash value and (b) it is computationally infeasible to find two inputs that map to the same hash value.
- 3.1.25 datagram TLS (DTLS):** A modification of TLS appropriate for packets transferred by UDP (or some other unreliable transport mechanism) as opposed to TCP (or some other reliable transport mechanism).
- 3.1.26 data plane:** A set of communication protocols defined to carry application and management data.
- 3.1.27 decode:** To parse an array of octets into a set of its constituent fields. *Contrast:* **decrypt, encode**.
- 3.1.28 decrypt:** To convert unreadable, encrypted data to readable, decrypted data using a decryption algorithm and a key. *Contrast:* **decode, encrypt**.
- 3.1.29 decryption algorithm:** An algorithm that takes as input ciphertext and a key and (if the correct key is provided) produces plaintext.

- 3.1.30 dedicated short range communication (DSRC):** A system for wireless communication between vehicles and the roadside and from one vehicle to another.
- 3.1.31 digital certificate:** A digitally signed document binding a public key to an identity and/or a set of permissions.
- 3.1.32 digital signature:** *See:* **public-key digital signature.**
- 3.1.33 elliptic curve cryptography (ECC):** A form of public-key cryptography based on the problem of finding discrete logarithms in a group defined over elliptic curves.
- 3.1.34 elliptic curve digital signature algorithm (ECDSA):** A digital signature algorithm based on the hard problems of elliptic curve cryptography.
- 3.1.35 elliptic curve integrated encryption scheme (ECIES):** A public-key encryption scheme based on the hard problems of elliptic curve cryptography.
- 3.1.36 encode:** To convert a data structure into an array of octets. *Contrast:* **decode, encrypt.**
- 3.1.37 encrypt:** To convert readable data to unreadable, encrypted data using an encryption algorithm and a key. *Contrast:* **decrypt, encode.**
- 3.1.38 encryption algorithm:** An algorithm that takes as input plaintext and a key and produces ciphertext.
- 3.1.39 endpoint:** The destination of a transmitted message. In the case of the IP stack, an endpoint is uniquely identified by the combination of IPv6 address and port number. In the case of the WSMP stack, an endpoint is uniquely identified by the Application Class ID and Application Context Mark.
- 3.1.40 expiry:** Removing a certificate from the system after a given time.
- 3.1.41 hardware security module (HSM):** A device that executes cryptographic functions in a secure execution environment, preventing unauthorized access to keys and functionality.
- 3.1.42 hash function:** *See:* **cryptographically secure hash function.**
- 3.1.43 hash value:** The output of a hash function.
- 3.1.44 implementation:** Any entity that sends messages over the WAVE stack. An application, or set of applications that use the same radio.
- 3.1.45 issuing certificate:** If the public key from certificate A can be used to verify the signature on certificate B, then A is the *issuing certificate* for B. *Contrast:* **subordinate certificate.**
- 3.1.46 keypair:** A private key and the corresponding public key for an asymmetric cryptosystem.
- 3.1.47 keystream:** A cryptographically secure stream of random data that is XORed with plaintext to produce ciphertext.
- 3.1.48 management plane:** The collection of functions performed in support of the communication functions provided by the data plane, but not directly involved in passing application data.
- 3.1.49 message integrity check (MIC):** A cryptographic checksum generated using a symmetric key.

- 3.1.50 message integrity check length (MIC length):** The length of the MIC generated by a MIC scheme.
- 3.1.51 message integrity check scheme (MIC scheme):** An instantiation of a cryptographic process that produces a MIC.
- 3.1.52 message signing certificate:** A certificate that is used to verify signatures on messages. *Contrast:* CA certificate.
- 3.1.53 networking services:** The collection of management plane and data plane functions at the network layer and transport layer.
- 3.1.54 on-board unit (OBU):** A WAVE device that can operate when in motion and supports information exchange with RSUs and other OBUs.
- 3.1.55 plaintext:** Data that has not been cryptographically processed (or has had all the cryptographic processing removed from it).
- 3.1.56 provider:** An entity in the system, typically though not exclusively an RSU, that offers services for consumption by entities known as users, which are typically though not exclusively on-board units (OBUs).
- 3.1.57 provider service table (PST):** The collection of data describing the applications that are registered with and available through a WAVE device.
- 3.1.58 public-key digital signature:** A code generated with an asymmetric cryptographic algorithm. The signing operation takes as input a message to be signed and the signer's private key, and produces a signature. The verification takes as input the purported signer's public key, the message, and the signature, and will produce "valid" if the signature was produced with the corresponding private key and the message and signature have not been altered since generation, and "invalid" otherwise.
- 3.1.59 public safety on-board unit (PSOBU):** An on-board unit (OBU) on a public safety vehicle.
- 3.1.60 public safety vehicle:** A vehicle that is given privileges over ordinary, private vehicles by a government agency to assist it in carrying out functions necessary for safety of life, or other public safety functions.
- 3.1.61 randomized encryption:** An encryption mechanism that has the property that the same message, encrypted twice with the same key, will give different results. Randomized encryption is semantically secure if an attacker who only sees the two ciphertexts will not be able to tell that they were derived from the same plaintext.
- 3.1.62 registration:** The provision of application parameters to a provider or user. These parameters are used by the provider to populate the PST, by the user to decide whether to link to a service advertised by a provider, and by both entities to manage the establishment and maintenance of the link in support of the service.
- 3.1.63 replay attack:** An attack in which an attacker records a valid message and retransmits it at a different time or location, hoping that listeners will still react to it as if it were valid.
- 3.1.64 revocation:** The act of removing a certificate from the system by distributing a signed statement that the certificate is no longer to be trusted.
- 3.1.65 revoked certificate store:** A unit's list of revoked certificates.

- 3.1.66 roadside unit (RSU):** A WAVE device that operates only when stationary and supports information exchange with on-board units (OBUs).
- 3.1.67 root certificate:** A certificate that was issued by itself and that can be used as a trust anchor to validate other certificates.
- 3.1.68 root certificate store:** A unit's store of root certificates.
- 3.1.69 security manager:** The application that is responsible for managing the root certificate store and the revoked certificate store on a unit.
- 3.1.70 semantically secure randomized encryption:** *See: randomized encryption.*
- 3.1.71 service:** An application offered by a provider for consumption by a user over an airlink.
- 3.1.72 service channel (SCH):** A secondary channel used for application specific information exchanges.
- 3.1.73 store:** A collection of data objects (typically certificates and CRLs in this standard) such that each object stays in the store until it expires, is superseded, or is revoked.
- 3.1.74 subordinate certificate:** If the public key from certificate A can be used to verify the signature on certificate B, then B is a *subordinate certificate* to A. *Contrast: issuing certificate.*
- 3.1.75 symmetric cryptographic algorithm:** A cryptographic algorithm that uses a single key. Anyone who knows a symmetric encryption key can both encrypt and decrypt. Anyone who knows a symmetric authentication key can both generate and check a MIC.
- 3.1.76 symmetric key:** *See: symmetric cryptographic algorithm.*
- 3.1.77 transactional application:** An application in which two entities exchange information. *Contrast: broadcast application.*
- 3.1.78 transport layer security (TLS):** An IETF standard providing for secure communications over TCP/IP.
- 3.1.79 trust anchor:** A known, trusted certificate. When a party receives a certificate, it will typically attempt to construct a certificate chain back to a trust anchor. If no trust anchor is available, the party shall use an out-of-band method to check that the certificate under consideration was issued by a legitimate issuer.
- 3.1.80 unit:** An implementation of a WAVE stack, with a single WAVE management entity (WME) and a single security manager.
- 3.1.81 vehicle host:** In-vehicle device running applications that communicate to external WAVE devices using the WAVE system through the OBU vehicle-host interface.
- 3.1.82 WAVE device:** A device that contains a WAVE-conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium (see IEEE Std 802.11 and IEEE P1609.4).
- 3.1.83 WAVE management entity (WME):** The set of management functions, defined in IEEE P1609.3, required to provide WAVE Networking Services; the management entity that performs these functions.

- 3.1.84 WAVE routing advertisement (WRA):** Network configuration information broadcast by the roadside unit (RSU).
- 3.1.85 WAVE service advertisement (WSA):** A collection of data collected by the WME and used to populate the WSIE.
- 3.1.86 WAVE service information element (WSIE):** A collection of configuration data transmitted by a provider (either OBU or RSU), which includes the Provider Service Table, and in the case of the RSU the WAVE Routing Advertisement, as well as security credentials.
- 3.1.87 WAVE short message (WSM):** A message sent over WSMP and routed to a receiving application using an ACID and an ACM rather than an IP address and port.
- 3.1.88 WAVE short message protocol (WSMP):** A protocol, defined in IEEE P1609.3, for routing WAVE short messages to receiving applications.
- 3.1.89 WAVE stack:** The network stack described in the IEEE 1609 series of standards.
- 3.1.90 wireless access in vehicular environment (WAVE):** A series of standards, developed by the IEEE P1609 working group, for vehicle-to-vehicle and vehicle-to-roadside wireless communications.

3.2 Abbreviations and acronyms

AES	Advanced Encryption Standard
ACID	application class identifier
ACM	application context mark
ASN.1	Abstract Syntax Notation 1
BER	Basic Encoding Rules (used with ASN.1)
CA	certificate authority
CBC	cipher block chaining
CCH	control channel
CCM	Counter with CBC MIC
CRL	certificate revocation list
CSR	certificate signing request
CSRSC	certificate signing request signing certificate
DIC	DSRC Industry Consortium
DNS	domain name system
DSRC	Dedicated Short Range Communications
DTLS	datagram TLS
ECC	elliptic curve cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
FEC	Forward Error Correction
FIPS	Federal Information Processing Standard
GPS	Global Positioning System
HSM	hardware security module
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv6	Internet Protocol version 6
ITS	Intelligent Transportation Systems
LLC	logical link control
MAC	medium access control
MIC	message integrity check
MIME	Multipurpose Internet Mail Extensions

MLME	MAC Layer Management Entity
MPDU	message protocol data unit
NIST	National Institute for Standards and Technology
NITSA	National Intelligent Transportation Systems Architecture
OBU	on-board unit
PDU	protocol data unit
PHY	Physical Layer
PLME	Physical Layer Management Entity
PSCRC	public safety certificate requesting certificate
PSOB	public safety on-board unit
PST	Provider Service Table
RFC	Request for Comments
RSU	roadside unit
SAE	Society of Automotive Engineers
S/MIME	secure MIME
SP	special publication
TLS	Transport Layer Security
UDP	User Datagram Protocol
UTF	Unicode Transformation Format
V2I	vehicle-to-infrastructure
V2V	vehicle-to-vehicle
WAVE	Wireless Access in Vehicular Environments
WBSS	WAVE Basic Service Set
WGS	World Geodetic System
WME	WAVE Management Entity
WRA	WAVE Routing Advertisement
WSA	WAVE Service Advertisement
WSIE	WAVE Service Information Element
WSM	WAVE Short Message
WSMP	WAVE Short Message Protocol
XOR	exclusive-OR

3.3 Terminology, applications, implementations, and the security manager

For an overview of the entire WAVE system and an introduction to cryptographic terminology, see Annex D.

The WAVE system involves many different types of entities. In day-to-day operations, the following entities are the most active:

Providers—Units that provide services on one or more service channels.

Users—Units that consume services offered by providers.

Roadside Units (RSUs)—WAVE devices that operate only when stationary and support information exchange with OBUs. RSUs are typically embedded in infrastructure elements such as road signs and traffic signals; they may be moved from one site to another but do not operate when in motion. Providers will usually be RSUs. RSUs are licensed by site and may provide services on one or more service channels.

On-Board Units (OBUs)—WAVE devices that can operate when in motion and support information exchange with RSUs and other OBUs. OBUs are typically embedded in vehicles, though they may also be portable (hand-held). OBUs operate in private vehicles, and a major goal of the security protocols in this document is to enable them to operate without violating a driver's reasonable assumptions about personal privacy. OBUs will usually be users, but may be providers in certain cases.

An important subset of OBUs consists of

Public Safety On-Board Units (PSOBUs)—Network nodes embedded in public safety vehicles. These nodes are permitted by the relevant authorities to operate particular public safety applications such as traffic signal prioritization.

In addition, the following entities support security services:

Certificate Authorities (CAs)—Entities that are able to authorize other entities via the issuance and revocation of certificates.

In this standard:

- An “endpoint” is the source or destination of any message. In the case of the IP stack, an endpoint is uniquely identified by the combination of IPv6 address and port number. In the case of the WSMP stack, an endpoint is uniquely identified by the Application Class ID and Application Context Mark, described in IEEE P1609.3.
- An “application” processes messages received at one or more endpoints, or sends messages to one or more endpoints. A receiving application may aggregate the messages sent to multiple endpoints. A sending application is identified with the endpoint to which it sends its messages.
- An “implementation” refers to any collection of applications on a single instance of the DSRC/WAVE radio stack.

The WME of IEEE P1609.3 is an application in this terminology, in that it processes management messages.

This standard specifies the behavior of a security manager application. This application has the responsibility for managing the root certificate and revoked certificate stores defined in 7.2.2 and 7.2.7. The security manager application receives messages over the WAVE stack. This standard does not specify any messages that are sent by the security manager. There is one security manager per WAVE stack. See 8.3 for a further description.

4. Presentation language

4.1 General

This standard specifies message formats using a presentation language based on the presentation language used by TLS (IETF RFC 2246 [B11]). This clause describes that presentation language and its encoding. A copyright statement for this clause may be found in Annex H.

In this standard, *encoding* is used to denote the process of converting from an internal representation of a structure to a flat octet string containing the same information. *Decoding* is used to denote the process of converting a flat octet string into an internal representation of the structure encoded in that octet string. The process of decoding can fail if the received octet string is not a valid encoding of the expected structure. In this case, the received octet string cannot be parsed.

4.2 Notation conventions

Elements of the presentation language are presented in `this font`. (Courier)

Presentation language statements contain variable names, data types, and functions. Variable names are all lowercase. Multiple words in a variable name are indicated by underscores, as in `variable_name`. Data

types begin with an uppercase letter and may contain a mixture of upper and lowercase. Multiple words in a data type are indicated by uppercase letters, as in `DataType`. The exceptions to this rule are the built-in data types `uint8`, `uint16`, `uint32`, `uint64`, and `opaque`. Functions (such as `select` statements) follow the same conventions as variable names.

For compactness, a field within a structure may be referred to using the `structure.field` notation.

Comments may be included preceded by `//`.

```
// this is a comment
```

4.3 Basic block size

The basic data block size is one octet. Multiple octet data items are concatenations of octets in network byte order. All lengths are expressed in octets.

4.4 Numbers

The following numeric types are predefined.

- `uint8` is a single-octet encoding of an unsigned 8-bit integer.
- `uint16` is a two-octet big-endian encoding of an unsigned 16-bit integer.
- `uint32` is a four-octet big-endian encoding of an unsigned 32-bit integer.
- `uint64` is an eight-octet big-endian encoding of an unsigned 64-bit integer.
- `sint32` is a four-octet encoding of a 32-bit signed integer, in which (as in BER encoding) the first bit is the sign bit and the remaining 31 bits encode the integer in big-endian form.

4.5 Fixed-length vectors

The syntax for specifying a new type `TNew` that is a fixed-length vector of type `TOld` is

```
TOld TNew[n];
```

where `n`, the length in octets, is a multiple of the size of `T`. The vector is encoded as `n` octets containing the data.

Example (1):

```
uint8 XY8[2];
```

This defines an `XY8` as a concatenation of `uint8`s of length 2 octets (which should not be read as “a concatenation of two `uint8`s”, even though in this case the two are equivalent.). An `XY8` encoding the values (0x12, 0x34) is encoded as the octets 12 34.

Example (2):

```
uint32 XY32[8];
```

This defines an `XY32` as a concatenation of `uint32`s of length 8 octets, in other words a concatenation of two `uint32`s. An `XY32` encoding the values (0x12, 0x34) is encoded as the octets 00 00 00 12 00 00 00 34.

4.6 Variable-length vectors

The syntax for specifying a new type TNew that is a variable-length vector of type TOld and maximum length n is

```
TOld  TNew<n>;
```

The vector is encoded in (length, value) form, where the length in octets is encoded as a big-endian integer of the minimum size necessary to encode n. In definitions of types with variable length vectors, n shall take one of the values 2^8-1 , $2^{16}-1$, $2^{32}-1$, or $2^{64}-1$, denoting that the length is encoded in one, two, four, or eight octets respectively.

Example (1):

```
uint8      AsciiChar;
AsciiChar  Name<2^8-1>;
```

This defines a name as a concatenation of AsciiChars of length no more than 2^8-1 octets. A name encoding the value “abc” is encoded as the octets 03 61 62 63. The leading octet, 03, is the length and the remainder is the data.

Example (2):

```
uint8      AsciiChar;
AsciiChar  LongName<2^16-1>;
```

This defines a long_name as a concatenation of AsciiChars of length no more than $2^{16}-1$ octets. A long_name encoding the value “abc” is encoded as the octets 00 03 61 62 63. The leading two octets, 00 03, are the length and the remainder is the data.

Example (3):

```
uint8      AsciiChar;
AsciiChar  Name<2^8-1>;
Name       OfficerNames<2^8-1>;
```

This defines the OfficerNames type as a concatenation of names of length no more than 2^8-1 octets. An OfficerNames encoding the two names “abc” and “def” is encoded as the octets 08 03 61 62 63 03 64 65 66.

NOTE—In this example, it is up to the implementer to ensure that the individual names are short enough for the concatenation of their encodings to fit in 2^8-1 or fewer octets.¹⁰

4.7 The opaque and opaqueExtLength type

The notation

```
opaque FieldName[n]
```

or

```
opaque FieldName<n>
```

¹⁰ Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

is used to denote a block of data that can be fixed or variable-length, exactly n bytes long in the fixed-length case (denoted with square brackets) or up to n bytes long in the variable-length case (denoted with angle brackets). Data in a field of type `opaque` is not further parsed by the security services. In this standard, `opaque` types carry application data that the security services protect.

The data in a variable-length field of type `opaque` is encoded preceded by its length, which shall be less than the value n . The value n shall take one of the values 2^8-1 , $2^{16}-1$, $2^{32}-1$, or $2^{64}-1$, denoting that the length is encoded in one, two four, or eight octets respectively.

The notation

```
opaqueExtLength FieldLength<n>
```

is used to denote a variable-length block of data that is not further parsed by the security services. The data in a field of type `opaqueExtLength` is *not* preceded by its length when encoded. Instead, the length of the field is deduced from external information.

4.8 Enumerated type

A field of type `enum` is a list of labels, each with a unique value, and an optional unlabelled maximum value. An `enum` can only assume the labeled values declared in the definition. An `enum` is encoded as an integer large enough to hold the maximum value given.

Example (1):

```
enum {usa(0), canada(1), mexico(2)}
    NAFTASignatories;
```

Because the maximum value given is 2, the `enum` is encoded as a single octet. The label `canada` as a member of `NAFTASignatories` is encoded as the value 01.

Example (2):

```
enum {usa(0), canada(1), mexico(2), (2^8-1)}
    NAFTASignatories;
```

Because the maximum value given is 2^8-1 , the `enum` is encoded as a single octet. The label `canada` as a member of `NAFTASignatories` is encoded as the value 01.

Example (3):

```
enum {usa(0), canada(1), mexico(2), (2^16-1)}
    NAFTASignatoriesWithRoomForExpansion;
```

Because the maximum value given is $2^{16}-1$, the `enum` is encoded as two octets. The label `canada` as a member of `NAFTASignatoriesWithRoomForExpansion` is encoded as the value 00 01.

An `enum` may note that certain fields are reserved for test purposes and shall not be used for operations, even in future versions of this standard. The notation for this is:

```
enum {usa(0), canada(1), mexico(2), reserved(240...255), (2^8-1)}
    NAFTASignatories;
```

4.9 Constructed types

Structured types may be constructed from primitive types for convenience using the `struct` syntax, as in the following example.

```
uint8      AsciiChar;
AsciiChar  Name<2^8-1>;
enum       {chair(0), treasurer(1), secretary(2)}
           OfficerPosition;

struct     {
           Name          name;
           OfficerPosition position;
           } Officer;
officer    officers<2^16-1>;
```

An organization with Alice as chair, Bob as treasurer, and Carol as secretary might encode its officers structure as follows (line breaks included for clarity only):

```
00 19          // length of encoded data is 19 octets
              // encoded as 2-octet integer
05            // length of "Alice"
41 6C 69 63 65 // "Alice", ASCII-encoded
00            // chair(0)
03            // length of "Bob"
42 6F 62      // "Bob", ASCII-encoded
01            // treasurer(1)
05            // length of "Carol"
43 61 72 6F 6C // "Carol", ASCII-encoded
02            // secretary(2)
```

4.10 The case statement

Defined structures may have variants based on some available knowledge. The selector shall be an enumerated type, as shown below, or a flag selector, discussed in 4.12. For example:

```
uint16      Pressure;
enum        {car(0), motorbike(1)}
           VehicleType;
struct      {
           VehicleType type;
           select (type) {
           case car:
               Pressure FrontLeft;
               Pressure FrontRight;
               Pressure RearLeft;
               Pressure RearRight;
           case motorbike:
               Pressure Front;
               Pressure Rear;
           }
           } TirePressureInfo;
```

The syntax

```
case (sel_1): ;
```

is used within a `select` statement to denote that one possible value of the selector (in this case, `sel_1`) has no additional fields associated with it. Alternatively, values of the selector that have no additional fields associated with them may be omitted from the `select` statement.

The syntax

```

case (sel_1):
case (sel_2):
    Type1 t1;
case (sel_3):
    Type2 t2;

```

is used within a `select` statement to denote that multiple values of the selector (in this case, `sel_1` and `sel_2`) have the same additional fields associated with them (in this case, `t1`). The general rule is that the first semicolon (;) appearing after the `case` statement indicates that the `case` statement is terminated by the next `case` statement if one exists, and by the closing brace (}) otherwise.

4.11 The extern statement

A structure may depend on information external to it. The notation is given below.

```

uint16      Pressure;
struct      {
    extern  VehicleType type;
    select (type) {
    case car:
        Pressure FrontLeft;
        Pressure FrontRight;
        Pressure RearLeft;
        Pressure RearRight;
    case motorbike:
        Pressure Front;
        Pressure Rear;
    }
} TirePressureInfo;

```

For those structures in this standard that use external information, the text explains where the external information is to be found.

4.12 Flags

4.12.1 Use of flags field

The flags field provides an alternative mechanism for constructing conditional structures. A `flags` type consists of a declaration of the different values that it may encode. The `if_set()` and `if_not_set()` statements are used to make the contents of a structure conditional on a flags type.

An example of the use of a flags field is

```

flags      {start_included(0), end_included(1)}
           TimeInfoFlags;
uint64      Time;
struct      {
    TimeInfoFlags  time_flags;
    if_set(time_flags, start_included) {
        Time64      start_time;
    }
    if_set(time_flags, end_included) {
        Time64      end_time;
    }
} TimeInfo;

```

The values of the entries in the `flags` declaration need not be consecutive. The maximum value of an entry in the `flags` declaration is 2047.

The `if_set` and `if_not_set` statements shall be used with opening and closing braces to denote the portions of the structure that are conditional on the given flag.

The `if_any_set` and `if_any_not_set` statements may take an arbitrary number of arguments. The first argument is the field to be checked and the remaining arguments are the values to check for. This is simply to allow compact representations, so that

```

        if_set(flags, x) {
            uint32      z;
        }
        if_set(flags, y) {
            uint32      z;
        }

```

may be represented as

```

        if_any_set(flags, x, y) {
            uint32      z;
        }

```

4.12.2 Encoding of flags field

An encoded flags field consists of a single octet, encoding its length in octets, followed by the flag data, which is a big-endian integer containing the flags. The `flags` field should be the minimum necessary length to encode all the flags that have been set.

A flag with value v is set in an encoded `flags` vector by ORing into the encoded vector the value $(1 \ll v)$, where “ $\ll$ ” denotes left-shifting. For example, a vector where flags 2 and 5 were set would be the octets 01 24, and a vector where the flag 8 was set would be the octets 02 01 00.

If no flags are set, the flags field is encoded as the single octet 00.

5. Secured Messages

5.1 General

This clause describes secure message formats for use in WAVE systems.

5.2 SecuredMessage type

Many of the specific applications discussed below make use of a single generic secured message format:

```

struct {
    uint8      protocol_version; // 1 for this version
    MessageType type;
    select (type) {
        case unsecured :
            opaque      message<2^32-1>;
        case signed :
            SignedMessage signed_message;
        case encrypted :
            EncryptedMessage encrypted_message;
    } SecuredMessage;
}

```

```
enum    {    unsecured (0), signed(1), encrypted (2),
            reserved (240...255), (28-1)
        }    MessageType;
```

The fields in the SecuredMessage have the following meanings.

- protocol_version contains the current version of the protocol. The version described in this document is version 1, represented by the integer 1. There are no major or minor version numbers.
- type contains the type of the message. This tells the receiving unit how to interpret the message body. This standard defines the types unsecured (0), which is included for use with WAVE short messages (WSMs); signed (1) indicating a signed message; and encrypted (2) indicating an encrypted message. Message types from 240 through 255 shall not be assigned and are reserved for private usage in test environments.

If the type field is signed, then the remaining octets are a SignedMessage, defined in 5.3. If the type field is encrypted, then the message block is an EncryptedMessage, defined in 5.13.

5.3 SignedMessage, ToBeSignedMessage, and MessageFlags types

```
struct {
    FullySpecifiedAppID    application;
    MessageFlags            mf;
    opaque                  application_data<216-1>;
    if_flag_set (mf, use_generation_time) {
        Time64              generation_time;
    }
    if_flag_set (mf, expires) {
        Time64              expiry_time;
    }
    if_flag_set (mf, use_location) {
        3DLocationAndConfidence    transmission_location;
    }
} ToBeSignedMessage;

struct {
    SignerInfo              signer;
    ToBeSignedMessage       unsigned_message;
    Signature               signature;
} SignedMessage;

flags {fragment(0), use_generation_time(1), expires(2),
        use_location(3)} MessageFlags;
```

In the ToBeSignedMessage structure:

- application identifies the application that will receive the SignedMessage. The ApplicationID type used here is discussed in more detail in 5.8. The application.type field shall take the value fully_specified in a SignedMessage.
- mf may contain the flags:
 - fragment(0) indicating that the message has been fragmented using the procedure specified in 8.5.
 - use_generation_time(1) indicating that the message will contain the transmission time.
 - expires(2) indicating that the message contains a lifetime field.
 - use_location(3) indicating that the transmission_location field shall be included.
- application_data contains the data for the application. Unless otherwise indicated, this is not interpreted by the security protocol and is passed unchanged up to the application.

- `generation_time` contains the time at which the message was generated. The `Time64` type is defined in 5.9. An implementation shall not send two messages with the same `generation_time` value.
- `expiry_time`, if present, contains the time at which the message expires.
- `transmission_location` contains the location of the generating unit at the time the message was generated. The `3DLocationAndConfidence` type is defined in 5.19.

In the `SignedMessage` structure:

- `signer` determines the keying material and hash algorithm used to sign the message. The `SignerInfo` type is defined in 5.10.
- `unsigned_message` contains the message data, as above.
- `signature` contains the digital signature itself. The `Signature` type is defined in 5.11.

The signature is calculated as follows: Given a signature algorithm that takes as inputs an octet string M , the message to be signed, and a private key Pr of the appropriate form for the signing algorithm, encode the `ToBeSignedMessage` as an octet string and use the resulting octet string as the input M to the signature algorithm.

The signature is verified as follows: Given a verification algorithm that takes as inputs an octet string M , the message to be verified; S , the signature on that message; and Pu , the public key to be used to verify the message, and given a `SignedMessage sm`: extract the encoded `ToBeSignedMessage` from the signed message and run the verification algorithm using the extracted `ToBeSignedMessage` as the input M , the signature from the `SignedMessage` as the input S , and the public key identified in the `SignerInfo` as the input Pu .

5.4 SignedWSM and ToBeSignedWSM types

```

struct {
    MessageFlags          mf;
    opaqueExtLength       wsm_payload<2^16-1>;
    if_flag_set (mf, use_generation_time) {
        Time64          generation_time;
    }
    if_flag_set (mf, expires) {
        Time64          expiry_time;
    }
    if_flag_set (mf, use_location) {
        3DLocationAndConfidence    transmission_location;
    }
} ToBeSignedWSM;

struct {
    SignerInfo          signer;
    ToBeSignedWSM       unsigned_wsm;
    Signature            signature;
} SignedWSM;

```

The `ToBeSignedWSM` type is a variant of the `ToBeSignedMessage` type. The fields have the same meanings as in the `ToBeSignedMessage` type, except that the `application` field is omitted.

The `SignedWSM` type is a variant of the `SignedMessage` type. The fields have the same meanings as in the `SignedMessage` type, except that the signed data is a `ToBeSignedWSM` structure rather than a `ToBeSignedMessage` structure.

The use of this type is described in 8.2.

5.5 PublicKey, PKAlgorithm, and SymmAlgorithm types

```

struct {
    PKAlgorithm          algorithm;
    select(algorithm) {
        case ecdsa_nistp224_with_sha224:
        case ecdsa_nistp256_with_sha256:
            ECPublicKey      public_key;
        case ecies_nistp256:
            SymmAlgorithm     supported_symm_algs<2^8-1>;
            ECPublicKey      public_key;
        }
    } PublicKey;

enum {
    ecdsa_nistp224_with_sha224 (0),
    ecdsa_nistp256_with_sha_256 (1), ecies_nistp256 (2),
    reserved (240..255), (2^8-1)
} PKAlgorithm;

enum {
    aes_128_ccm (0), reserved (240..255), (2^8-1) }
SymmAlgorithm;

```

The `PublicKey` structure encodes a public key and states with what algorithm the public key shall be used.

All implementations of this standard shall support the signing algorithm ECDSA over the two NIST curves p224 and p256, specified with the `PKAlgorithm` values `ecdsa_nistp224_with_sha224` and `ecdsa_nistp256_with_sha256`.

The only supported asymmetric encryption algorithm in this version of this standard is ECIES. All implementations of this standard that support encryption shall support this algorithm over the NIST curve p256, specified with the `PKAlgorithm` value `ecies_nistp256`.

- The `supported_symm_algs` field lists the symmetric algorithms that may be used with the ECIES public key. The only symmetric algorithm currently supported is AES in CCM mode as described in NIST SP 800-38C (see the references in Clause 2), denoted by `aes_128_ccm(0)`. All implementations that support encryption shall support this algorithm.
- `public_key` contains the encoded public key. The `ECPublicKey` type is specified in 5.6.

Additional public-key encryption or signature algorithms may be supported by reserving a `PKAlgorithm` value for them. Values 240-255 shall not be assigned and are reserved for private usage in test environments.

Additional symmetric algorithms may be supported by reserving a `SymmAlgorithm` value for them. Values 240-255 shall not be assigned and are reserved for private usage in test environments.

5.6 ECPublicKey type

An ECDSA or ECIES public key is specified as follows:

```

struct {
    extern uint8    point_size;
    opaque          point[point_size];
} ECPublicKey;

```

ECC points are expressed in the LSB-compressed format of IEEE Std 1363-2000. The value `point_size` depends on the `PKAlgorithm` associated with the key. If the `PKAlgorithm` is `ecdsa_nistp224_with_sha224`, `point_size` shall be 29. If the `PKAlgorithm` is `ecdsa_nistp256_with_sha_256` or `ecies_nistp256`, `point_size` shall be 33.

5.7 CertID8 and CertID10 type

```
opaque CertID8[8];
opaque CertID10[10];
```

The `CertID8` and `CertID10` types are used to identify a certificate. The `CertIDX` for a given certificate, where `X` is 8 or 10, shall be calculated by calculating the SHA-256 hash of the certificate and taking the low-order `X` bytes of the hash output.

5.8 The ApplicationID and FullySpecifiedAppID types

```
struct {
    AIDType    type;
    select (type) {
        case fully_specified:
            uint8    acid;
            opaque    acm<2^8-1>;
        case match_any_acm:
            uint8    acid;
        case from_issuer: ;
    }
} ApplicationID;

enum {fully_specified(0), match_any_acm(1), from_issuer(2),
      (2^8-1)} AIDType;

struct {
    uint8    acid;
    opaque    acm<2^8-1>;
} FullySpecifiedAppID;
```

These types are used by the security services to determine whether a message may legitimately address a given application. This is carried out by checking that the `FullySpecifiedAppID` in a signed message is consistent with the `ApplicationID` in the signer's certificate. The process of checking consistency is covered in detail in 7.3.3.5.

In an `ApplicationID`:

- `type` shall take the value `fully_specified(0)`, `match_any_acm(1)` or `from_issuer(1)`. This field indicates which of the other fields are present in the encoded structure.
- `acid`, if present, contains the application class identifier. This field shall only take the values given in IEEE P1609.3.
- `acm`, if present, contains additional information about the receiving application. This data may be used by the security layer to check consistency as specified in 7.3.3.5.

In a `FullySpecifiedAppID`:

- `acid` contains the application class identifier. This field shall only take the values given in IEEE P1609.3.
- `acm` contains additional information about the receiving application. This data may be used by the security layer to check consistency as specified in 7.3.3.5.

NOTE—If the ACID and ACM are present, encoding them following the encoding rules of this document gives the same results as encoding them with the rules presented in IEEE P1609.3.

5.9 Time64 and Time32 types

```
uint64    Time64;
uint32    Time32;
```

This standard supports two time types to allow bandwidth to be saved if a low-precision time is sufficient.

The `Time64` type is a 64-bit integer, encoded in big-endian format, giving the number of microseconds since 00:00:00 UTC, 1 January, 2004.

The `Time32` type is a 32-bit integer, encoded in big-endian format, giving the number of seconds since 00:00:00 UTC, 1 January, 2004.

Subclause 7.3.3.2 discusses the source of the time measurements.

5.10 SignerInfo type

```
struct {
    SignerIdentifierType type;
    select (type) {
        case certificate:
            WAVECertificate certificate;
        case certificate_digest:
            CertID8 digest;
        case certificate_chain:
            WAVECertificate certificates<2^16-1>;
        case self: ;
    }
} SignerInfo;

enum {certificate(0), certificate_digest(1), certificate_chain (2),
      self (3), (2^8-1)} SignerIdentifierType;
```

The `SignerInfo` structure allows the recipient of a message to determine which keying material to use to authenticate the message. It may contain either a certificate, a message digest of a certificate, or a certificate chain, as follows:

- If the `type` field contains `certificate(0)` then the `SignerInfo` contains a certificate.
- If the `type` field contains `certificate_digest(1)` then the `SignerInfo` contains the `CertID8` corresponding to the relevant certificate. This is the low-order 8 octets of the SHA-256 hash of that certificate.
- If the `type` field contains `certificate_chain(2)` then the `SignerInfo` contains multiple certificates. The first certificate shall be the certificate that signed the message. The other certificates may appear in any order. A compliant application shall support certificate chains containing five certificates, and may support chains that contain more. For more details on certificate chains see 7.3.3.4.

If the `type` field contains `self(3)` then the message is signed by a key contained within itself. This type may only be used to sign a certificate request. All other messages shall be signed with one of the other types.

5.11 Signature type

```

struct {
    extern PKAlgorithm algorithm;
    select(algorithm) {
        case ecdsa_nistp224_with_sha224:
        case ecdsa_nistp256_with_sha256:
        case ecdsa_with_sha256:
            ECDSASignature ecdsa_signature;
    }
} Signature;

```

The signature field contains the actual signature. In order to parse it, a recipient needs to know the algorithm that it was created with. In the case of a signed message (or CRL) the algorithm is to be found in the `unsigned_certificate.public_key.algorithm` field of the message (or CRL) signing certificate. In the case of a certificate the algorithm is to be found in the `unsigned_certificate.public_key.algorithm` field of the issuing certificate. See 5.15 for the definition of `unsigned_certificate`.

5.12 ECDSASignature type

```

struct {
    extern uint8 curve_order_octets;
    opaque      r[curve_order_octets];
    opaque      s[curve_order_octets];
} ECDSASignature;

```

ECDSA signatures are performed as described in FIPS 186-2. The input message is the encoded `unsigned_message` value. An ECDSA signature consists of two values r and s . These integers shall be converted into octet strings of the same length as the curve order n using the procedure of FIPS 186-2. The strings are padded with leading zeros as necessary to obtain the appropriate length. The strings are then concatenated to form the appropriate signature value.

The value `curve_order_octets` is constant for a given `NamedCurve` `PKAlgorithm` value. It takes the value given in Table 1.

Table 1—curve_order_octets for different values of PKAlgorithm

NamedCurve	curve order octets
nistp224	28
nistp256	32

5.13 EncryptedMessage, EncryptedContentInfo, and RecipientInfo types

```

struct {
    EncryptedContentType content_type;
    SymmAlgorithm         symm_algorithm;
    RecipientInfo         recipients<2^16-1>;
    select(symm_algorithm) {
        case aes_128_ccm:
            AESCCMCiphertext ciphertext;
    }
} EncryptedMessage;

struct {
    CertID8 cert_id;
    extern PKAlgorithm pk_encryption;
    select (pk_encryption) {
        case ecies_nistp256:
            ECIESNISTp256EncryptedKey enc_key;
    }
}

```

```

    }
  } RecipientInfo;

  enum {app_data(0), signed(1), (2^8-1)} EncryptedContentType;

```

The `EncryptedMessage` type supports encrypting a message to one or more recipients using the recipients' public keys. The message is encrypted with a fresh symmetric key generated by the sender, and the symmetric key is then encrypted for each recipient separately using that recipient's public key. See 7.4 for more details.

The `EncryptedMessage` type contains the following fields:

- `content_type` takes the values `app_data` or `signed`. If it is `app_data`, the contents of the encrypted message are application-specific. If it is `signed`, the contents of the encrypted message are a signed message and shall require further processing by the security services before they can be passed to the application.
- `symm_algorithm` contains an identifier for the symmetric algorithm that was used to encrypt the message. The `SymmAlgorithm` type is defined in 5.5. This field is provided primarily for future expansion; The only supported value in this version of the standard is `aes_128_ccm`, indicating that the message was encrypted with AES using a 128-bit key in CCM mode. The `SymmAlgorithm` value here shall also appear in the `supported_symm_algorithms` field of all of the certificates identified in the `cert_id` fields of the recipients.
- `recipients` contains one or more `RecipientInfos`, defined below.
- The `application_data_ciphertext` field contains the encrypted message. The general procedure to be used in generating the encrypted message is discussed given in 7.4.2, and the general procedure for processing received encrypted messages is discussed given in 7.4.3. The specific procedure for encryption and decryption with `aes_128_ccm` is discussed in 7.4.2.2.

The `RecipientInfo` type is used to transfer the symmetric key to each recipient separately. It contains the following fields:

- `cert_id` contains the low-order eight octets of the SHA-256 hash of the recipient's certificate. This shall be the certificate that contains the public key used to encrypt the symmetric key.
- `pk_algorithm` is set equal to the algorithm field of the encryption key in the referenced certificate. See 5.17 for more information.
- The final field contains the encrypted key. The format of this field depends on `pk_algorithm`. This standard only supports one `pk_algorithm`, `ecies_nistp256`, and only one format for the encrypted key, the `ECIESNISTp256EncryptedKey` type.

5.14 ECIESNISTp256EncryptedKey and AESCCMCiphertext types

```

struct {
    extern uint32      symm_key_len;
    opaque             v[33];
    opaque             c[symm_key_len];
    opaque             t[16];
} ECIESNISTp256EncryptedKey;

struct {
    opaque             nonce[12];
    opaque             ccm_ciphertext<2^16-1>;
} AESCCMCiphertext;

```