ETSI TS 103 097 V1.1.1 (2013-04)

Technical Specification

Intelligent Transport Systems (ITS);
Security;
Security header and certificate formats

Reference
DTS/ITS-0050023

Keywords
ITS, privacy, protocol, security

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: "*Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards*", which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Intelligent Transport Systems (ITS).

Introduction

Security mechanisms for ITS consist of a number of parts. An important part for interoperability is a common format for data elements being transferred between ITS stations for security purposes.

The present document intends to provide such a format definition. A special focus is to include as much as possible from existing standards. At the same time, the major goal is simplicity and extensibility of data structures.

1 Scope

The present document specifies security header and certificate formats for Intelligent Transport Systems. These formats are defined specifically for securing G5 communication.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] IEEE Std. 1363-2000: "Standard Specifications For Public Key Cryptography".

[2] NIMA Technical Report TR8350.2: "Department of Defense World Geodetic System 1984. Its Definition and Relationships with Local Geodetic Systems".

[3] ISO 3166-1: "Codes for the representation of names of countries and their subdivisions - Part 1: Country codes".

[4] NIST SP 800-38C: "Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality".

[5] IETF RFC 2246: "The TLS Protocol Version 1.0".

[6] ETSI TS 102 637-2: "Intelligent Transport Systems (ITS);Vehicular Communications;Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service".

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] IEEE Std 1363a-2004: "Standard Specifications For Public Key Cryptography- Amendment 1: Additional Techniques".

[i.2] IEEE Std. 1609.2-2012 (draft D12): "Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages".

[i.3] IEEE Std. 1609.2-2012 (draft D17): "Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages".

[i.4] IEEE Std. 1609.3-2010: "Wireless Access in Vehicular Environments (WAVE) - Networking Services".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

enumeration: set of values with distinct meaning

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

CA	Certificate Authority
CAM	Cooperative Awareness Message
CRL	Certificate Revocation List
DENM	Decentralized Environmental Notification Message
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
G5	5,9 GHz radio communications
ITS	Intelligent Transport Systems
ITS-AID	ITS Application ID
ITS-S	Intelligent Transport Systems Station
NIMA	National Imagery and Mapping Agency
NIST SP	National Institute of Standards and Technology, Special Publication
PSID	Provider Service Identifier

NOTE: It is a synonym for ITS-AID.

SSP	Service Specific Permissions
TAI	Temps Atomique International (International Atomic Time)
UTC	Universal Time Coordinated
WGS	World Geodetic System

4 Basic format elements

4.1 Presentation Language

The presentation language is derived from the Internet Engineering Task Force (IETF) RFC 2246 (TLS) [5] and from IEEE Std. 1609.2-2012 [i.2] (draft D12) and is described in table 1.

NOTE: The presentation language is not formally defined. Parsing tools based on this notation cannot be guaranteed to be consistent or complete.

Table 1: Presentation language

Element	Description	Example(s)
Variable names	Variable names are given in lower case	variable_name
Basic data types	Basic data types are given in lower case	uint8, uint16, uint32, uint64
Composed data types	Composed data types are given with at least the first letter in upper case	MyDataType
Comments	Comments start with the "//" indicator	// This is a comment
Numbers	Numbers are given as signed or unsigned big-endian octets, i.e. network byte order	uint8, uint16, uint32, uint64, sint32

Fixed-length vectors	Fixed-length vectors have a data type and a fixed octet size given in square brackets	uint8 Coordinates[2];
		// two uint8 values
		uint32 Coordinates[8];
		// two uint32 values

Variable-length vectors with fixed length encoding	The number in angle brackets gives the maximum number of octets. Depending on the maximum size, the first 1, 2, 4 or 8 bytes encode the actual field length	uint8 AsciiChar; AsciiChar Name<2^8-1>; // "abc" encoded as // 0x03, 0x61, 0x62, 0x63 AsciiChar LongName<2^16-1>; // "abc" encoded as // 0x00, 0x03, 0x61, 0x62, 0x63

Variable-length vectors with variable-length encoding	<var> indicates variable-length encoding. The length itself is encoded with a number of "1" bits according to the additional number of octets used to encode the length, followed by a "0" bit and the actual length value.	uint8 AsciiChar;
		AsciiChar Name<var>;
		
		// encoding examples: (the bits with
		// grey background represent the
		// length encoding of the vector's
		// length, X the first of the //
		vector's following payload bits)
		
		// Vector length 5:
		// Bits: 00000101 XXXXXXXX XXXXXXXX
		
		// Vector length 123:
		// Bits: 01111011 XXXXXXXX XXXXXXXX
		
		// Vector length 388:
		// Bits: 10000001 10000100 XXXXXXXX

Opaque fields	Opaque fields are blocks of data whose content interpretation is not further specified	opaque fieldname[n]; opaque fieldname<n>; opaque fieldname<var>;

Enumerations	Enumerations are list of labels with a unique value for each label, and optionally a maximum value (which then determines length of encoding)	enum {de(0), fr(1), it(2)} Country;
		enum {de(0), fr(1), it(2), (2^8-1)}
		Country;
		// both variants encoding in one
		// octet
		enum {de(0), fr(1), it(2), (2^16-1)}
		Country;
		// Encoding in two octets

Constructed types	Constructed types contain other types	struct { Name name; Country country; } Person;

Case statements	Case statements are used inside constructed types to change the contents of the constructed type depending on the value of the variable given in brackets	struct {
		Name name;
		Country country;
		select(country) {
		case de:
		uint8 age;
		case fr:
		AsciiChar given_name<2^8-1>;
		}
		} Person;

External data	This is external data that has impact on a struct, e.g. in a select statement. It shall be described from where the external data is obtained.	struct {
		Name name;
		extern Country country;
		select(country) {
		case de:
		uint8 age;
		case fr:
		AsciiChar given_name<2^8-1>;
		}
		} Person;

4.2 Specification of basic format elements

4.2.1 IntX

IntX int_x;

This data type encodes an integer of variable length. The length of this integer is encoded by a number of 1 bit followed by a 0 bit, where the number of 1 bit is equal to the number of additional octets used to encode the integer besides those used (partially) to encode the length.

EXAMPLE: 00001010 encodes the integer 10, while 10001000 10001000 encodes the integer 2 184. The bits encoding the length of the element are colored with a grey background.

NOTE: This definition is similar to the definition of PSID in IEEE 1609.3-2010 [i.4], clause 8.1.3, but allows bigger values of the encoded integer.

4.2.2 PublicKeyAlgorithm

```
enum {
    ecdsa_nistp256_with_sha256(0),
    ecies_nistp256(1),
    reserved(240..255),
    (2^8-1)
} PublicKeyAlgorithm;
```

This enumeration lists supported algorithms based on public key cryptography. Values in the range of 240 to 255 shall not be used as they are reserved for internal testing purposes.

NOTE: This definition is similar to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.16, but `ecdsa_nistp224_with_sha224` is not supported by the present document. As a consequence, the numbering of identical elements (e.g. `ecdsa_nistp256`) differs.

4.2.3 SymmetricAlgorithm

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

This enumeration lists supported algorithms based on symmetric key cryptography. Values in the range of 240 to 255 shall not be used as they are reserved for internal testing purposes. The algorithm `aes_128_ccm` denotes the symmetric key cryptography algorithm AES-CCM as specified in NIST SP 800-38C [4].

NOTE: Except naming, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.23.

4.2.4 PublicKey

```
struct {
    PublicKeyAlgorithm      algorithm;
    select(algorithm) {
    case ecdsa_nistp256_with_sha256:
        EccPoint        public_key;
    case ecies_nistp256:
        SymmetricAlgorithm  supported_symm_alg;
        EccPoint        public_key;
    unknown:
        opaque              other_key<var>;
    }
} PublicKey;
```

This structure defines a wrapper for public keys by specifying the used algorithm and - depending on the value of algorithm - the necessary data fields:

- ecdsa_nistp256_with_sha256: the specific details regarding ECC contained in an EccPoint structure shall be given.
- ecies_nistp256: the specific details regarding ECC contained in an EccPoint structure and the symmetric key algorithm contained in a SymmetricAlgorithm structure shall be given.
- unknown: in all other cases, a variable-length vector containing opaque data shall be given.

NOTE: Except naming of included types, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.31.

4.2.5 EccPoint

```
struct {
    extern PublicKeyAlgorithm   algorithm;
    extern uint8                field_size;
    EccPointType                type;
    opaque                      x[field_size];
    select(type) {
    case x_coordinate_only:
    case compressed_lsb_y_0:
    case compressed_lsb_y_1:
                                ;
    case uncompressed:
        opaque                  y[field_size];
    unknown:
        opaque                  data<var>;
    }
} EccPoint;
```

This structure defines a public key based on elliptic curve cryptography according to IEEE Std 1363-2000 [1] clause 5.5.6. An EccPoint encodes a coordinate on a two dimensional elliptic curve. The x coordinate of this point shall be encoded in x as an unsigned integer in network byte order. Depending on the key type, the y coordinate shall be encoded case-specific:

- x_coordinate_only: only the x coordinate is encoded, no additional data shall be given.
- compressed_lsb_y_0: the point is compressed and y's least significant bit is zero, no additional data shall be given.
- compressed_lsb_y_1: the point is compressed and y's least significant bit is one, no additional data shall be given.
- uncompressed: the y coordinate is encoded in the field y. The y coordinate contained in a vector of length field_size containing opaque data shall be given.
- unknown: in all other cases, a variable-length vector containing opaque data shall be given.

The uint8 field_size defining the lengths of the vectors containing the raw keys shall be derived from the given algorithm and the mapping as defined in table 2. The necessary algorithm shall be given as an external link to the parameter pk_encryption specified in the structure RecipientInfo.

Table 2: Derivation of field sizes depending on the used algorithm

PublicKeyAlgorithm value	Length in octets
ecdsa_nistp256_with_sha256	32

NOTE: Except inclusion of all remaining elements of the enumeration EccPointTypethat previously matched to case uncompressed and inclusion of case unknown, this definition is identical to the EccPublicKey in IEEE 1609.2 Draft D12 [i.2], clause 6.2.18.

4.2.6 EccPointType

```
enum {
    x_coordinate_only(0),
    compressed_lsb_y_0(2),
    compressed_lsb_y_1(3),
    uncompressed(4),
    (2^8-1)
} EccPointType;
```

This enumeration lists supported ECC key types.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.19.

4.2.7 EncryptionParameters

```
struct {
    SymmetricAlgorithm  symm_algorithm;
    select(symm_algorithm) {
    case aes_128_ccm:
        opaque          nonce[12];
    unknown:
        opaque          params<var>;
    }
} EncryptionParameters;
```

This structure holds basic parameters and additional data required for encryption and decryption of data using different symmetric encryption algorithms. In case of `aes_128_ccm` a 12 octet nonce shall be given. In other cases the data shall be given as a variable-length vector containing `opaque` data. It is out of scope of this definition how resulting ciphertexts are transported. Typically, a ciphertext should be put into a `Payload` data structure marked as `encrypted` using the `PayloadType`.

NOTE: This structure is not available in IEEE 1609.2 Draft D12 [i.2].

4.2.8 CrlSeries

```
uint32  CrlSeries;
```

This number identifies a CRL series. The definition of a CRL series itself is outside the scope of the present document.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.21.

4.2.9 Signature

```
struct {
    PublicKeyAlgorithm  algorithm;
    select(algorithm) {
    case ecdsa_nistp256_with_sha256:
        EcdsaSignature  ecdsa_signature;
    unknown:
        opaque          signature<var>;
    }
} Signature;
```

This structure defines a container that encapsulates signatures based on public key cryptography. Depending on the value of `algorithm`, different data structures define the algorithm-specific details:

- `ecdsa_nistp256_with_sha256`: the signature contained in an `EcdsaSignature` structure shall be given.
- `unknown`: in all other cases, a variable-length vector containing the signature as `opaque` data shall be given.

The data in this structure can be used to verify a data structure's integrity. In conjunction with a matching SignerInfo structure, the data structure's authenticity can also be verified.

It is necessary to note the following bullet points:

- Clause 5.6 defines which parts of a SecuredMessage data structure are covered by a signature.
- The length of the security_field<var> variable length vector in the SecuredMessage containing the Signature field shall be calculated before creating the signature using the length of the signature.
- Before calculating the actual signature, the length field of the surrounding variable length vector TrailerField shall be calculated using the value of field_size, since this length field is part of the signed content.

NOTE: Except naming and full inclusion (not marked as extern) of the enumeration PublicKeyAlgorithm, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.15.

4.2.10 EcdsaSignature

```
struct {
    extern PublicKeyAlgorithm   algorithm;
    extern uint8                field_size;
    EccPoint                    R;
    opaque                      s[field_size];
} EcdsaSignature;
```

This structure defines the details needed to describe an ECDSA based signature. The field s contains the signature. This field's length field_size is derived from the applied ECDSA algorithm using the mapping as specified in table 2. The extern link that specifies the algorithm points to the algorithm defined in the surrounding Signature structure. R contains the associated ECC public key.

NOTE: Except naming of included type PublicKeyAlgorithm, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.17.

4.2.11 SignerInfo

```
struct {
    SignerInfoType type;
    select(type){
    case self:
                            ;
    case certificate_digest_with_ecdsap256:
        HashedId8           digest;
    case certificate:
        Certificate         certificate;
    case certificate_chain:
        Certificate         certificates<var>;
    case certificate_digest_with_other_algorithm:
        PublicKeyAlgorithm  algorithm;
        HashedId8           digest;
    unknown:
        opaque              info<var>;
    }
} SignerInfo;
```

This structure defines how to give information about the signer of a message. The included cryptographic identity can be used in conjunction with the structure Signature to verify a message's authenticity. Depending on the value of type, the SignerInfo's data fields shall contain the following entries:

- self: the data is self-signed. Therefore, no additional data shall be given. This shall only be used in case of a certificate request.

- certificate_digest_with_ecdsap256: an 8 octet digest of the relevant certificate contained in a HashedId8 structure shall be given.
- certificate: the relevant certificate itself contained in a Certificate structure shall be given.
- certificate_chain: a complete certificate chain contained in a variable-length vector of type Certificate shall be given. The last element of the chain shall contain the certificate used to sign the message, the next to last element shall contain the certificate of the CA that signed the last certificate and so on. The first element of the chain needs not be a root certificate.
- certificate_digest_with_other_algorithm: an 8 octet digest contained in a HashedId8 structure and the corresponding public key algorithm contained in a PublicKeyAlgorithm structure shall be given.
- unknown: in all other cases, a variable-length vector containing information as opaque data shall be given.

NOTE: Except naming, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.4.

4.2.12 SignerInfoType

```
enum {
    self(0),
    certificate_digest_with_ecdsap256(1),
    certificate(2),
    certificate_chain(3),
    certificate_digest_with_other_algorithm(4),
    reserved(240..255),
    (2^8-1)
} SignerInfoType;
```

This enumeration lists methods to describe a message's signer. Values in the range of 240 to 255 shall not be used as they are reserved for internal testing purposes.

NOTE: This definition is similar to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.5, but naming and certificate_digest_with_ecdsap224 is not supported by the present document. As a consequence, the numbering of identical elements (e.g. certificate_chain) differs.

4.2.13 HashedId8

```
opaque  HashedId8[8];
```

This value is used to identify data such as a certificate. It shall be calculated by first computing the SHA-256 hash of the input data, and then taking the least significant eight bytes from the hash output.

NOTE: Except naming, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.6.

4.2.14 HashedId3

```
opaque  HashedId3[3];
```

This value is used to give an indication on an identifier, where real identification is not required. This can be used to request a certificate from other surrounding stations. It shall be calculated by first computing the SHA-256 hash of the input data, and then taking the least significant three bytes from the hash output. If a corresponding HashedId8 value is available, it can be calculated by truncating the longer HashedId8 to the least significant three bytes.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

4.2.15 Time32

```
uint32  Time32;
```

Time32 is an unsigned 32-bit integer, encoded in big-endian format, giving the number of International Atomic Time (TAI) seconds since 00:00:00 UTC, 1 January, 2010.

NOTE 1: The period of 2^{32} seconds lasts about 136 years, that is until 2146.

NOTE 2: Except change of the epoch (starting with 2010 instead of 2004), this definition is identical to the one in IEEE 1609.2 Draft D17 [i.3], clause 6.3.31.

4.2.16 Time64

```
uint64  Time64;
```

Time64 is a 64-bit unsigned integer, encoded in big-endian format, giving the number of International Atomic Time (TAI) microseconds since 00:00:00 UTC, 1 January, 2010.

NOTE: Except change of the epoch (starting with 2010 instead of 2004, this definition is identical to the one in IEEE 1609.2 Draft D17 [i.3], clause 6.2.12.

4.2.17 Time64WithStandardDeviation

```
struct {
    Time64  time;
    uint8   log_std_dev;
} Time64WithStandardDeviation;
```

This structure defines how to encode time along with the standard deviation of time values. log_std_dev values 0 to 253 represent the rounded up value of the log to the base 1,134666 of the implementation's estimate of the standard deviation in units of nanoseconds. The value 254 represents any value greater than $1{,}134666^{244}$ nanoseconds, i.e. a day or longer. The value 255 indicates that the standard deviation is not known.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D17 [i.3], clause 6.2.11.

4.2.18 Duration

```
uint16  Duration;
```

This uint16 encodes the duration of a time span (e.g. a certificate's validity). The first three bits shall encode the units as given in table 3. The remaining 13 bits shall be treated as an integer encoded in network byte order.

NOTE: Except naming, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.5.

Table 3: Interpretation of duration unit bits

Bits	Interpretation
000	seconds
001	minutes (60 seconds)
010	hours (3 600 seconds)
011	60 hour blocks (216 000 seconds)
100	years (31 556 925 seconds)
101, 110, 111	undefined

4.2.19 TwoDLocation

```
struct {
    sint32  latitude;
    sint32  longitude;
} TwoDLocation;
```

This structure defines how to specify a two dimensional location. It is used to define validity regions of a certificate. `latitude` and `longitude` encode a coordinate in tenths of micro degrees relative to the World Geodetic System (WGS)-84 datum as defined in NIMA Technical Report TR8350.2 [2].

The permitted values of `latitude` range from -900 000 000 to +900 000 000. The value 900 000 001 shall indicate the latitude as not being available.

The permitted values of `longitude` range from -1 800 000 000 to +1 800 000 000. The value 1 800 000 001 shall indicate the longitude as not being available.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.18.

4.2.20 ThreeDLocation

```
struct {
    sint32  latitude;
    sint32  longitude;
    opaque  elevation[2];
} ThreeDLocation;
```

This structure defines how to specify a three dimensional location. `latitude` and `longitude` encode coordinate in tenths of micro degrees relative to the World Geodetic System (WGS)-84 datum as defined in NIMA Technical Report TR8350.2 [2].

The permitted values of `latitude` range from -900 000 000 to +900 000 000. The value 900 000 001 shall indicate the latitude as not being available.

The permitted values of `longitude` range from -1 800 000 000 to +1 800 000 000. The value 1 800 000 001 shall indicate the longitude as not being available.

`elevation` shall contain the elevation relative to the WGS-84 ellipsoid in decimeters. The value is interpreted as an asymmetric signed integer with an encoding as follows:

- 0x0000 to 0xEFFF: positive numbers with a range from 0 to +6 143,9 meters. All numbers above +6 143,9 are also represented by 0xEFFF.
- 0xF001 to 0xFFFF: negative numbers with a range from -409,5 to -0,1 meters. All numbers below -409,5 are also represented by 0xF001.
- 0xF000: an unknown elevation.

EXAMPLES: 0x0000 = 0 meters

0x03E8 = 100 meters

0xF7D1 = -209,5 meters (0xF001 + 0x07D0 = -409,5 meters + 200 meters)

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.12.

4.2.21 GeographicRegion

```
struct {
    RegionType              region_type;
    select(region_type){
    case circle:
        CircularRegion      circular_region;
    case rectangle:
        RectangularRegion   rectangular_region<var>;
    case polygon:
        PolygonalRegion     polygonal_region;
    case id:
        IdentifiedRegion    id_region;
    case none:
                            ;
    unknown:
        opaque              other_region<var>;
    }
} GeographicRegion;
```

This structure defines how to encode geographic regions. These regions can be used to limit the validity of certificates.

In case of `rectangle`, the region shall consist of a variable-length vector of rectangles that may be overlapping or disjoint. The variable-length vector shall not contain more than 6 rectangles. The region covered by the rectangles shall be continuous and shall not contain holes.

NOTE: Except inclusion of case `id`, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.13.

4.2.22 RegionType

```
enum {
    none(0),
    circle(1),
    rectangle(2),
    polygon(3),
    id(4),
    reserved(240..255),
    (2^8-1)
} RegionType;
```

This enumeration lists possible region types. Values in the range of 240 to 255 shall not be used as they are reserved for internal testing purposes.

NOTE: This definition is similar to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.14, but the identifier numbering differs, the region ID `id` was added and `from_issuer` removed.

4.2.23 CircularRegion

```
struct {
    TwoDLocation    center;
    uint16          radius;
} CircularRegion;
```

This structure defines a circular region with `radius` given in meters and center at `center`. The region shall include all points on the reference ellipsoid's surface with a distance smaller or equal than the radius to the center point.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.15.

4.2.24 RectangularRegion

```
struct {
    TwoDLocation    northwest;
    TwoDLocation    southeast;
} RectangularRegion;
```

This structure defines a rectangular region with the uppermost, leftmost point at `northwest` and the rightmost, lowest point at `southeast`.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.16.

4.2.25 PolygonalRegion

```
TwoDLocation    PolygonalRegion<var>;
```

This variable-length vector describes a region by enumerating points on the region's boundary. The points shall be linked to each other, with the last point linked to the first. No intersections shall occur and no more than 12 points shall be given. The specified region shall be continuous and shall not contain holes.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.17.

4.2.26 IdentifiedRegion

```
struct {
    RegionDictionary    region_dictionary;
    Int16               region_identifier;
    IntX                local_region;
} IdentifiedRegion;
```

This structure defines a predefined geographic region determined by the region dictionary `region_dictionary` and the region identifier `region_identifier`. `local_region` may optionally specify a more detailed region within the region. If the whole region is meant, `local_region` shall be set to 0.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

4.2.27 RegionDictionary

```
enum {
    iso_3166_1(0),
    un_stats(1),
    (2^8-1)
} RegionDictionary;
```

This enumeration lists dictionaries containing two-octet records of globally defined regions. The dictionary that corresponds to `iso_3166_1` shall contain values that correspond to numeric country codes as defined in ISO 3166-1 [3]. The dictionary that corresponds to `un_stats` shall contain values as defined by the United Nations Statistics Division, which is a superset of ISO 3166-1 [3] including compositions of regions.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5 Specification of security header

5.1 SecuredMessage

```
struct {
    uint8           protocol_version;
    uint8           security_profile;
    HeaderField     header_fields<var>;
    Payload         payload_fields<var>;
    TrailerField    trailer_fields<var>;
} SecuredMessage;
```

This structure defines how to encode a generic secured message:

- `protocol_version` specifies the applied protocol version. For compliance with the present document, protocol version `1` shall be used.
- `security_profile` specifies the security profile for this secured message. The profiles define the contents of the variable header, payload and trailer fields. A message that does not conform to the profile is invalid. The default value shall be set to `0`, if no specific profile is used.
- `header_fields` is a variable-length vector that contains multiple information fields of interest to the security layer. If not defined otherwise in a message profile, the sequence of header fields shall be encoded in ascending numerical order of their type value.
- `payload_fields` is a variable-length vector containing the message's payload. Multiple payload types in one message are allowed.
- `trailer_fields` is a variable-length vector containing information after the payload, for example, necessary to verify the message's authenticity and integrity. If not defined otherwise in a message profile, the sequence of trailer fields shall be encoded in ascending numerical order of the type value.

Further information about how to fill these variable-length vectors is given via security profiles in clause 7.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5.2 Payload

```
struct {
    PayloadType type;
    select (type) {
    case signed_external:
                ;
    case unsecured:
    case signed:
    case encrypted:
    case signed_and_encrypted:
    unknown:
        opaque  data<var>;
    }
} Payload;
```

This structure defines how to encode payload. In case of externally signed payload, no payload data shall be given as all data is external. In this case, the signature shall be contained in the trailer fields. In all other cases, the data shall be given as a variable-length vector containing `opaque` data.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5.3 PayloadType

```
enum  {
    unsecured(0),
    signed(1),
    encrypted(2),
    signed_external(3),
    signed_and_encrypted(4),
    (2^8-1)
} PayloadType;
```

This enumeration lists the supported types of payloads.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5.4 HeaderField

```
struct {
    HeaderFieldType type;
    select(type) {
    case generation_time:
        Time64                          generation_time;
    case generation_time_standard_deviation:
        Time64WithStandardDeviation     generation_time_with_standard_deviation;
    case expiration:
        Time32                          expiry_time;
    case generation_location:
        ThreeDLocation                  generation_location;
    case request_unrecognized_certificate:
        HashedId3                       digests<var>;
    case message_type:
        uint16                          message_type;
    case signer_info:
        SignerInfo                      signer;
    case recipient_info:
        RecipientInfo                   recipients<var>;
    case encryption_parameters:
        EncryptionParameters            enc_params;
    unknown:
        opaque                          other_header<var>;
    }
} HeaderField;
```

This structure defines how to encode information of interest to the security layer. Its content depends on the value of `type`:

- `generation_time`: the point in time this message was generated contained in a `Time64` structure shall be given.
- `generation_time_standard_deviation`: the point in time this message was generated with additional confidence described by the standard deviation of the time value contained in a `Time64WithStandardDeviation` structure shall be given.
- `expiration`: the point in time the validity of this message expires contained in a `Time32` structure shall be given.
- `generation_location`: the location where this message was created contained in a `ThreeDLocation` structure shall be given.
- `request_unrecognized_certificate`: a request for certificates shall be given in case that a certificate from a peer has not been transmitted before. This request consists of a variable-length vector of 3 octet long certificate digests contained in a `HashedId3` structure to identify the requested certificates.
- `message_type`: the type of the message shall be given. These types are specified in the security profiles in clause 7.

Furthermore, the HeaderField structure defines cryptographic information that is required for single-pass processing of the payload:

- `signer_info`: information about the message's signer contained in a `SignerInfo` structure shall be given. If present, the SignerInfo structure shall come first in the array of HeaderFields, unless this is explicitly overridden by the security profile.

- `encryption_parameters`: additional parameters necessary for encryption purposes contained in an `EncryptionParameters` structure shall be given.

- `recipient_info`: information specific for certain recipients (e.g. data encrypted with a recipients public key) contained in a variable-length vector of type `RecipientInfo` shall be given.

For extensibility, the structure contains a variable field:

- `unknown`: in all other cases, a variable-length vector containing `opaque` data shall be given.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5.5 HeaderFieldType

```
enum {
    generation_time(0),
    generation_time_confidence(1),
    expiration(2),
    generation_location(3),
    request_unrecognized_certificate(4),
    message_type(5),
    signer_info(128),recipient_info(129),
    encryption_parameters(130),
    (2^8-1)
} HeaderFieldType;
```

This enumeration lists the supported types of header fields.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5.6 TrailerField

```
struct {
    TrailerFieldType            type;
    select(type) {
    case signature:
        Signature               signature;
    unkown:
        opaque                  security_field<var>;
    }
} TrailerField;
```

This structure defines how to encode information used by the security layer after processing the payload. A trailer field may contain data of the following cases:

- `signature`: the signature of this message contained in a `Signature` structure shall be given. The signature is calculated over the hash of the encoding of all previous fields (`version`, `header_fields` field and the `payload_fields` field), including the encoding of their length.

  If there is a `payload` field with `type` equal to `signed_external`, the data shall be included in the hash calculation immediately after the `payload` field, encoded as an `opaque<var>`, i.e. as if it was included.

  If there is one or more `payload` field whose `type` does not contain the keyword "signed" (`unsecured` or `encrypted`), the length fields shall be included, the corresponding `data` fields shall be excluded from the hash calculation.

If further trailer fields are included in a SecuredMessage, the signature structure shall include all fields in the sequence before, and exclude all fields in the sequence after the signature structure, if not otherwise defined via security profiles.

- unknown: in all other cases, a variable-length vector containing opaque data shall be given.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5.7 TrailerFieldType

```
enum {
    signature(1),
    (2^8-1)
} TrailerFieldType;
```

This enumeration lists the supported types of trailer fields.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

5.8 RecipientInfo

```
struct {
    HashedId8                       cert_id;
    PublicKeyAlgorithm              pk_encryption;
    select (pk_encryption) {
    case ecies_nistp256:
        EciesNistP256EncryptedKey   enc_key;
    unknown:
        opaque                      enc_key<var>;
    }
} RecipientInfo;
```

This structure contains information for a message's recipient. This information is used to distribute recipient-specific data. cert_id determines the 8 octet identifier for the recipient's certificate. Depending on the value of pk_encryption, the following additional data shall be given:

- ecies_nistp256: an encrypted key contained in an EciesNistP256EncryptedKey structure shall be given.
- unknown: in all other cases, a variable-length vector containing opaque data encoding an encrypted key shall be given.

NOTE: Except naming of included type PublicKeyAlgorithm and full inclusion of pk_encryption (not extern), this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.24.

5.9 EciesNistP256EncryptedKey

```
 struct {
    extern SymmetricAlgorithm   symm_alg;
    extern uint32               symm_key_len;
    EccPoint                    v;
    opaque                      c[symm_key_len];
    opaque                      t[20];
} EciesNistP256EncryptedKey;
```

This structure defines how to transmit an EciesNistP256-encrypted symmetric key as defined in IEEE Std 1363a-2004 [i.1]. The `EccPoint v` contains the sender's ECC key used for the Elliptic Curve Encryption Scheme. The vector `c` contains the encrypted public key. The vector `t` contains the output tag. The `symm_key_len` defining the length of vector `c` containing the raw key shall be derived from the given algorithm `symm_alg` and the mapping as defined in table 4. The necessary algorithm shall be given as an external link to the parameter `symm_algorithm` specified in the structure `EncryptionParameters`. To ensure the external link to the `SymmetricAlgorithm symm_alg` can be resolved, this `EciesNistP256EncryptedKey` structure shall be preceded by an according `EncryptionParameters` structure.

Table 4: Derivation of symmetric key size depending on the used algorithm

PublicKeyAlgorithm value	Length in octets
aes_128_ccm	16

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.25.

6 Specification of certificate format

6.1 Certificate

```
struct {
    uint8                   version;
    SignerInfo              signer_info<var>;
    SubjectInfo             subject_info;
    SubjectAttribute        subject_attributes<var>;
    ValidityRestriction     validity_restrictions<var>;
    Signature               signature;
} Certificate;
```

This structure defines how to encode a certificate.

- `version` specifies this certificate's version and shall be set to `1` for conformance with the present document.
- Information on this certificate's signer is given in the variable-length vector `signer_info`.
- `subject_info` specifies information on this certificate's subject.
- Further information on the subject is given in the variable-length vector `subject_attributes`. The elements in the `subject_attributes` array shall be encoded in ascending numerical order of their `type` value, unless this is specifically overridden by a security profile. `subject_attributes` shall not contain two entries with the same `type` value.
- The variable-length vector `validity_restrictions` specifies restrictions regarding this certificate's validity. . The elements in the `validity_restrictions` array shall be encoded in ascending numerical order of their `type` value, unless this is specifically overridden by a security profile. `validity_restrictions` shall not contain two entries with the same `type` value.
- `signature` holds the signature of this certificate signed by the responsible CA. The signature shall be calculated over the encoding of all preceding fields, including all encoded lengths. If the subject_attributes field contains a field of type `reconstruction_value`, the `signature` field shall be omitted.

NOTE 1: A certificate is considered valid if the current time is within the validity period specified in the certificate, the current region is within the validity region specified in the certificate, the type of the certificate is valid for the current type of communication, the signature, which covers all fields except the signature itself, is valid, and the certificate of the signer is valid as signer for the given certificate's type. If the certificate is self-signed, it is valid if it is stored as a trusted certificate.

NOTE 2: This definition differs substantially from the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.1.

6.2 SubjectInfo

```
struct {
    SubjectType     subject_type;
    opaque          subject_name<2^8-1>;
} SubjectInfo;
```

This structure defines how to encode information about a certificate's subject. It contains the type of information in `subject_type` and the information itself in the variable-length vector `subject_name`. The `subject_name` variable-length vector shall have a maximum length of 32 bytes.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

6.3 SubjectType

```
enum {
    enrollment_credential(0),
    authorization_ticket(1),
    authorization_authority(2),
    enrollment_authority(3),
    root_ca(4),
    crl_signer(5),
    (2^8-1)
} SubjectType;
```

This enumeration lists the possible types of subjects:

- Regular ITS stations shall use certificates containing a `SubjectInfo` of `SubjectType` `enrollment_credential` when communicating with Enrollment CAs. Such certificates shall not be accepted as signers of other certificates or in regular communication by other ITS-Stations.
- Regular ITS stations shall use certificates containing a `SubjectInfo` of `SubjectType` `authorization_ticket` when communicating with other ITS-Stations. Such certificates shall not be accepted as signers of other certificates.
- Authorization CAs, which sign authorization tickets (pseudonyms) for ITS stations, shall use the `SubjectType` `authorization_authority`.
- Enrollment CAs, which sign enrollment credentials (long term certificates) for ITS stations, shall use the `SubjectType` `enrollment_authority`.
- Root CAs, which sign certificates of other CAs, shall use the `SubjectType` `root_ca`.
- Certificate revocation list signers shall use `SubjectType` `crl_signer`.

NOTE: This definition substantially differs from the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.3.

6.4 SubjectAttribute

```
struct {
    SubjectAttributeType    type;
    select(type) {
    case verification_key:
    case encryption_key:
        PublicKey           key;
    case reconstruction_value:
        EccPoint            rv;
    case assurance_level:
        SubjectAssurance    assurance_level;
    case its_aid_list:
        IntX                its_aid_list<var>;
    case its_aid_ssp_list:
        ItsAidSsp           its_aid_ssp_list<var>;
    case priority_its_aid_list:
        ItsAidPriority      its_aid_priority_list<var>;
```

```
    case priority_ssp_list:
        ItsAidPrioritySsp   its_aid_priority_ssp_list<var>;
    unknown:
        opaque              other_attribute<var>;
    }
} SubjectAttribute;
```

This structure defines how to encode a subject attribute. These attributes serve the purpose of specifying the technical details of a certificate's subject. Depending on the value of `type`, the following additional data shall be given:

- `verification_key` and `encryption_key`: a public key contained in a `PublicKey` structure shall be given.
- `reconstruction_value`: an ECC point contained in a `EccPoint` structure shall be given.
- `assurance_level`: the assurance level for the subject contained in a `SubjectAssurance` structure shall be given.
- `its_aid_list`: ITS-AIDs contained in a variable-length vector of type `IntX` shall be given.
- `Its_aid_ssp_list`: ITS-AIDs with associated SSPs contained in a variable-length vector of type `ItsAidSsp` shall be given.
- `priority_its_aid_list`: ITS-AIDs and associated maximum priorities contained in a variable-length vector of type `ItsAidPriority` shall be given.
- `priority_ssp_list`: ITS-AIDs and associated SSPs and maximum priorities contained in a variable-length vector of type `ItsAidPrioritySsp` shall be given.
- `unknown`: in all other cases, a variable-length vector containing `opaque` data shall be given.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

6.5 SubjectAttributeType

```
enum {
    verification_key(0),
    encryption_key(1),
    assurance_level(2),
    reconstruction_value(3),
    its_aid_list(32),
    its_aid_ssp_list(33),
    priority_its_aid_list(34),
    priority_ssp_list(35),
    (2^8-1)
} SubjectAttributeType;
```

This enumeration lists the possible types of subject attributes.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

6.6 SubjectAssurance

```
opaque  SubjectAssurance;
```

This field contains the ITS-S's assurance, which denotes the ITS-S's security of both the platform and storage of secret keys as well as the confidence in this assessment.

This field shall be encoded as defined in table 5, where "A" denotes bit fields specifying an assurance level, "R" reserved bit fields and "C" bit fields specifying the confidence.

Table 5: Bitwise encoding of subject assurance

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

In table 5, bit number 0 denotes the least significant bit. Bits 7 to 5 denote the ITS-S's assurance levels, bits 4 to 2 are reserved for future use and the bits 1 and 0 denote the confidence.

The specification of these assurance levels as well as the encoding of the confidence levels is outside the scope of the present document. The default (no assurance) shall be all bits set to 0.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

6.7 ValidityRestriction

```
struct {
    ValidityRestrictionType type;
    select(type){
    case time_end:
        Time32              end_validity;
    case time_start_and_end:
        Time32              start_validity;
        Time32              end_validity;
    case time_start_and_duration:
        Time32              start_validity;
        Duration            duration;
    case region:
        GeographicRegion    region;
    unknown:
        opaque              data<var>;
    }
} ValidityRestriction;
```

This structure defines ways to restrict the validity of a certificate depending on the value of `type`:

- `time_end`: the expiration date for the associated certificate contained in a `Time32` structure shall be given.
- `time_start_and_end`: the beginning of the validity contained in a `Time32` structure and the expiration date contained in another `Time32` structure shall be given.
- `time_start_and_duration`: the beginning of the validity contained in a `Time32` structure and the duration of validity contained in a `Duration` structure shall be given.
- `region`: the region the certificate is valid in contained in a `GeographicRegion` structure shall be given.
- `unknown`: in all other cases, a variable-length vector containing `opaque` data shall be given.

A valid certificate shall contain exactly one validity restriction of type `time_end`, `time_start_and_end`, and `time_start_and_duration`.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

6.8 ValidityRestrictionType

```
enum {
    time_end(0),
    time_start_and_end(1),
    time_start_and_duration(2),
    region(3),
    (2^8-1)
} ValidityRestrictionType;
```

This enumeration lists the possible types of restrictions to a certificate's validity.

NOTE: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

6.9 ItsAidSsp

```
struct {
    IntX        its_aid;
    opaque      service_specific_permissions<var>;
} ItsAidSsp;
```

This structure defines how to encode an ITS-AID with associated Service Specific Permissions (SSP). `service_specific_permissions` shall have a maximum length of 31 octets. The definition of SSPs is out of scope of the present document.

NOTE: This definition is similar to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.24, but uses different naming, a slightly more flexible encoding of the ITS-AID.

6.10 ItsAidPriority

```
struct {
    IntX        its_aid;
    uint8       max_priority;
} ItsAidPriority;
```

This structure defines how to encode an ITS-AID with an associated maximum priority. The priority defines an order for processing of different messages. Higher numbers equal higher priority.

NOTE: This definition is similar to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.12, but uses different naming and a slightly more flexible encoding of the ITS-AID.

6.11 ItsAidPrioritySsp

```
struct {
    IntX        its_aid;
    uint8       max_priority;
    opaque      service_specific_permissions<var>;
} ItsAidPrioritySsp;
```

This structure is a combination of `ItsAidSsp` and `ItsAidPriority`. It defines how an ITS-AID is associated with its specific permission set and maximum priority of CA certificates. `service_specific_permissions` shall have a maximum length of 31 octets. The definition of SSPs is out of scope for the present document.

NOTE: This definition is similar to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.29, but uses different naming, a slightly more flexible encoding of the ITS-AID.

7 Security profiles

7.1 Security profile for CAMs

This clause defines which fields shall be included in the SecuredMessage structure for Cooperative Awareness Messages (CAMs) as well as the scope of application of cryptographic features applied to the header.

The `security_profile` value for this profile shall be set to `1`.

These HeaderField elements shall be included in all CAMs. With the exception of signer_info, which is encoded first, all header_field elemenrts shall be included in ascending order according to the numbering of the enumeration of the according type structure:

- signer_info: this field shall contain exactly one field of the types: certificate_digest_with_ecdsap256 or certificate, according to the following rules:
    - In the normal case, the signer_info field of type certificate_digest_with_ecdsap256 shall be included.
    - Instead of including a field of type certificate_digest_with_ecdsap256, a signer_info field of type certificate shall be included one second after the last inclusion of a field of type certificate.
    - If the ITS-S receives a CAM from a previously unknown other ITS-S, it shall include a field of type certificate immediately in its next CAM, instead of including a field of type certificate_digest_with_ecdsap256. In this case, the timer for the next inclusion of a field of type certificate shall be restarted.
    - If an ITS-S receives a CAM whose security header includes a HeaderField of type request_unrecognized_certificate, then the ITS-S shall evaluate the list of HashedId3 digests included in that field. If the ITS-S finds a HashedId3 of its own, currently used certificate chain in that list, it shall include a signer_info field of type certificate immediately in its next CAM, instead of including a signer_info field of type certificate_digest_with_ecdsap256.
- generation_time: this field shall contain the current absolute time.
- message_type: this field shall be set to 2 (as defined in the ItsPduHeader in TS 102 637-2 [6]).

The HeaderField element request_unrecognized_certificate shall be included if an ITS-S received CAMs from other ITS-Ss, which the ITS-S has never encountered before and which included only a signer_info field of type certificate_digest_with_ecdsap256 instead of a signer_info TrailerField of type certificate. In this case, the signature of the received CAMs cannot be verified because the verification key is missing. The field digests<var> in the structure of request_unrecognized_certificate shall be filled with a list of HashedId3 elements of the missing ITS-S certificates.

NOTE: HashedId3 elements can be formed by using the least significant three bytes of the corresponding HashedId8.

generation_time_with_confidence shall not be used. None of the possible HeaderField cases shall be included more than once. Additional HeaderField types are allowed.

At least one Payload element shall be included in all CAMs.

These TrailerField elements shall be included in all CAMs:

- signature: this field shall contain a signature calculated over these fields of the SecuredMessage data structure:
    - protocol_version.
    - The variable-length vector header_fields including its length.
    - The length of the variable-length vector payload_fields, all included PayloadType fields and all payloads with a type that contains the word "signed". If the payload is marked as external, its contents shall be included in the hash as well, at the position where a non-external payload would be.
    - The length of the variable-length vector trailer_fields and all data preceding the signature, including the length of the signature field.

CAMs shall not be encrypted.

7.2 Security profile for DENMs

This clause defines which fields shall always be included in the SecuredMessage structure for Decentralized Environmental Notification Messages (DENMs) as well as the scope of application of cryptographic features applied to the header.

The `security_profile` value for this profile shall be set to 2.

These `HeaderField` elements shall be included in all DENMs. With the exception of signer_info, which is encoded first, all `header_field` elements shall be included in ascending order according to the numbering of the enumeration of the according type structure:

- `signer_info`: this field shall contain an element of type `certificate`.
- `generation_time`: this field shall contain the current absolute time.
- `generation_location`: this field shall contain the current location of the ITS-S at the time of generation of the DENM.
- `message_type`: this field shall be set to `1`.

`generation_time_with_confidence` shall not be used. None of the possible `HeaderField` cases shall be included more than once. Additional HeaderField types are allowed.

At least one `Payload` element shall be included in all DENMs.

These `TrailerField` elements shall be included in all DENMs:

- `signature`: this field shall contain a signature calculated over these fields of the `SecuredMessage` data structure:
  - protocol_version.
  - The variable-length vector header_fields including its length.
  - The length of the variable-length vector payload_fields, all included PayloadType fields and all payloads with a type that contains the word "signed". If the payload is marked as external, its contents shall be included in the hash as well, at the position where a non-external payload would be.
  - The length of the variable-length vector trailer_fields and all data preceding the signature, including the length of the signature field.

DENMs shall not be encrypted.

7.3 Generic security profile for other signed messages

This clause defines which fields shall always be included in the SecuredMessage structure for other signed messages as well as the scope of application of cryptographic features applied to the header.

The `security_profile` value for this profile shall be set to `3`.

These `HeaderField` elements shall be included. With the exception of signer_info, which is encoded first, all `header_field` elements shall be included in ascending order according to the numbering of the enumeration of the according type structure:

- `signer_info`: this field shall contain an element of type `certificate`.
- `generation_time`: this field shall contain the current absolute time.
- `generation_location`: this field shall contain the current location of the ITS-S at the time of generation of the DENM.

None of the possible `HeaderField` cases shall be included more than once. Additional HeaderField types are allowed.

At least one `Payload` element of type `signed`, `signed_external` or `signed_and_encrypted` shall be included.

These `TrailerField` elements shall be included:

- `signature`: this field shall contain a signature calculated over these fields of the `SecuredMessage` data structure:
  - protocol_version.
  - The variable-length vector header_fields including its length.
  - The length of the variable-length vector payload_fields, all included PayloadType fields and all payloads with a type that contains the word "signed". If the payload is marked as external, its contents shall be included in the hash as well, at the position where a non-external payload would be.
  - The length of the variable-length vector trailer_fields and all data preceding the signature, including the length of the signature field.

7.4 Profiles for certificates

This clause defines which types of variable fields shall always be included in certificates.

These `SubjectAttribute` elements shall be included:

- `verification_key:` this field shall contain the verification key of the sender that is used to create message signatures.
- `assurance_level:` this field shall contain the assurance level of the sender.

Exactly one of the following `ValidityRestriction` fields shall be included:

- `time_end:` this field shall contain the end of validity of the certificate.
- `time_start_and_end:` this field shall contain the validity period of the certificate.
- `time_start_and_duration:` this field shall contain the validity period of the certificate.

The options `time_start_and_end` or `time_start_and_duration` should be preferred.

For the field `signer_info`, exactly one of the following types shall be included:

- `certificate_digest_with_ecdsap256`
- `certificate`
- `certificate_chain`
- `certificate_digest_with_other_algorithm`

Apart from these fields, certificate contents may be extended depending on the purpose of the certificate.

7.4.1 Authorization tickets (pseudonymous certificates)

This clause defines additional aspects of authorization tickets (i.e. pseudonymous certificates).

These `SubjectAttribute` elements shall be included in addition to those specified in clause 7.4 for all certificates:

- `its_aid_list`: this field shall contain a list of ITS-AIDs

As ValidityRestriction field restricting the time of validity, time_start_and_end shall be included.

The SubjectInfo field of the authorization ticket shall be set to these values:

- subject_type: this field shall be set to authorization_ticket(1).
- subject_name: this field shall be set to 0x00 (empty name field).

7.4.2 Enrollment credential (long-term certificates)

This clause defines additional aspects of enrollment credentials (i.e. long-term certificates).

These SubjectAttribute elements shall be included in addition to those specified in clause 7.4 for all certificates:

- its_aid_list: this field shall contain a list of ITS-AIDs.

As ValidityRestriction field restricting the time of validity, time_start_and_end shall be included.

For the field signer_info, exactly one of the following types shall be included:

- certificate_digest_with_ecdsap256
- certificate
- certificate_chain

In the SubjectInfo field of the enrollment credential, subject_type shall be set to enrollment_credential(0).

7.4.3 Certificate authority certificates

This clause defines additional aspects of certificate authority certificates.

These SubjectAttribute elements shall be included in addition to those specified in clause 7.4 for all certificates:

- its_aid_list: this field shall contain a list of ITS-AIDs.

As ValidityRestriction field restricting the time of validity, time_start_and_end shall be included.

In the SubjectInfo field of the enrollment credential, subject_type shall be set to one of these types:

- authorization_authority(2), for authorization authorities, i.e. certificate authorities issuing authorization tickets.
- enrollment_authority(3), for enrollment authorities, i.e. certificate authorities issuing enrollment credentials.
- root_ca(4), for root certificate authorities.

The following SignerInfo fields shall be included:

- For root certificate authority certificates, the signer_info field shall be set to self(0).
- For other certificate authorities, the signer_info field shall be set to certificate(2).

Annex A (informative): Data structure examples

A.1 Example security envelope structure for CAM

The following structure shown in table A.1 is an example security header for a CAM message. The header transports the generation time, identifies the payload as signed, and includes the hash of a certificate, that is, no full certificate is included in this case. Finally, an ECDSA-NISTP-256 based signature is attached.

Table A.1: An example signed header for CAM

Element	Value	Description	Length in octets
SecuredMessage			
uint8 protocol_version	0x01		1
uint8 security_profile	0x01	CAM security profile value = 1	1
HeaderField header_fields<var>	0x16	length: 22 octets	1
HeaderFieldType type	0x80	signer_info	1
SignerInfoType signer_info	0x01	certificate_digest_with_ecdsap256	1
HashedId8 digest	[…]		8
HeaderFieldType type	0x00	generation_time	1
Time64 generation_time	[…]		8
HeaderFieldType type	0x05	message_type	1
unit16 message_type	0x0002	CAM	2
Payload payload_fields<var>	0x02	length: 2 octets	1
PaylodType payload_type	0x01	signed	1
opaque data<var>	0x00	length: 0 octets	1
[raw payload data]			0
TrailerField trailer_fields<var>	0x43	length: 67 octets	1
TrailerFieldType type	0x01	signature	1
PublicKeyAlgorithm algorithm	0x00	ecdsa_nistp256_with_sha_256	1
EcdsaSignature ecdsa_signature			
EccPoint R			
EccPointType type	0x01	compressed_lsb_y_0	2
opaque x[32]	[…]		32
opaque s[32]	[…]		32
The total size of the security header structure is 96 octets.			

A.2 Example structure of a certificate

The following structure shown in table A.2 is an example of a certificate.

Table A.2: An example structure of a certificate

Element	Value	Description	Length in octets
Certificate			
uint8 version	0x01		1
SignerInfo singner_info<var>	0x09	length: 9 octets	1
SignerInfoType type	0x01	certificate_digest_with_ecdsap256	1
HashedId8 digest	[…]		8
SubjectInfo subject_info			
SubjectType type	0x01	authorization_ticket	1
opaque subject_name<var>	0x00	length: 0 → no name	1
[subject name]			0
SubjectAttribute subject_attributes<var>	0x2b	length: 43	1
SubjectAttributeType type	0x00	verification_key	1
PublicKey key			
PublicKeyAlgorithm algorithm	0x00	ecdsa_nistp256_with_sha256	1
EccPoint public_key			
EccPointType type	0x01	compressed_lsb_y_0	2
opaque x[32]	[…]		32
SubjectAttributeType type	0x02	assurance_level	1
SubjectAssurance assurance_level	0x04	level_4	1
SubjectAttributeType type	0x33	its_aid_ssp_list	1
ItsAidSsp its_aid_ssp_list<var>	0x04	length: 4 octets	1
IntX its_aid	[…]		1
opaque service_specific_permissions<var>	0x02	length: 2 octets	1
[service specific permissions]	[…]		2
ValidityRestriction validity_restrictions<var>	0x09	length: 9 octets	1
ValidityRestrictionType type	0x01	time_start_and_end	1
Time32 start_validity	[…]		4
Time32 end_validity	[…]		4
Signature signature			
PublicKeyAlgorithm algorithm	0x00	ecdsa_nistp256_with_sha256	1
EcdsaSignature ecdsa_signature			
EccPoint R			
EccPointType type	0x01	compressed_lsb_y_0	2
opaque x[32]	[…]		32
opaque s[32]	[…]		32
The total size of this certificate is 133 octets.			

History

Document history		
V1.1.1	April 2013	Publication