ETSI TS 103 097 V1.2.1 (2015-06)

TECHNICAL SPECIFICATION

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

Reference

RTS/ITS-00531

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

Copyright Notification

Contents

Intellectual Property Rights ... 5
Foreword ... 5
Modal verbs terminology ... 5
Introduction ... 5
1 Scope ... 6
2 References ... 6
2.1 Normative references ... 6
2.2 Informative references ... 6
3 Definitions and abbreviations ... 7
3.1 Definitions ... 7
3.2 Abbreviations ... 7
4 Basic format elements ... 7
4.1 Presentation Language ... 7
4.2 Specification of basic format elements ... 9
4.2.1 IntX ... 9
4.2.2 PublicKeyAlgorithm ... 9
4.2.3 SymmetricAlgorithm ... 9
4.2.4 PublicKey ... 9
4.2.5 EccPoint ... 10
4.2.6 EccPointType ... 11
4.2.7 EncryptionParameters ... 11
4.2.8 Signature ... 11
4.2.9 EcdsaSignature ... 12
4.2.10 SignerInfo ... 12
4.2.11 SignerInfoType ... 13
4.2.12 HashedId8 ... 13
4.2.13 HashedId3 ... 13
4.2.14 Time32 ... 14
4.2.15 Time64 ... 14
4.2.16 Time64WithStandardDeviation ... 14
4.2.17 Duration ... 14
4.2.18 TwoDLocation ... 15
4.2.19 ThreeDLocation ... 15
4.2.20 GeographicRegion ... 15
4.2.21 RegionType ... 16
4.2.22 CircularRegion ... 16
4.2.23 RectangularRegion ... 16
4.2.24 PolygonalRegion ... 17
4.2.25 IdentifiedRegion ... 17
4.2.26 RegionDictionary ... 17
5 Specification of security header ... 17
5.1 SecuredMessage ... 17
5.2 Payload ... 18
5.3 PayloadType ... 18
5.4 HeaderField ... 18
5.5 HeaderFieldType ... 20
5.6 TrailerField ... 20
5.7 TrailerFieldType ... 20
5.8 RecipientInfo ... 21
5.9 EciesEncryptedKey ... 21
6 Specification of certificate format ... 22
6.1 Certificate ... 22
6.2 SubjectInfo ... 23

6.3 SubjectType 23
6.4 SubjectAttribute 23
6.5 SubjectAttributeType 24
6.6 SubjectAssurance 24
6.7 ValidityRestriction 25
6.8 ValidityRestrictionType 25
6.9 ItsAidSsp 25

7 Security profiles 26
7.1 Security profile for CAMs 26
7.2 Security profile for DENMs 27
7.3 Generic security profile for other signed messages 28
7.4 Profiles for certificates 29
7.4.1 Introduction 29
7.4.2 Authorization tickets (pseudonymous certificates) 30
7.4.3 Enrolment credential (long-term certificates) 30
7.4.4 Certificate authority certificates 30

Annex A (informative): Data structure examples 32

A.1 Example security envelope structure for CAM 32

A.2 Example structure of a certificate 33

Annex B (informative): Usage of ITS-AID and SSPs 34

History 35

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).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

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

2.1 Normative 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 reference 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.

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

[1] IEEE™ 1363-2000: "IEEE 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 102 940: "Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management".

[7] ETSI TS 102 965 (V1.2.1): "Intelligent Transport Systems (ITS); Application Object Identifier (ITS-AID); Registration".

2.2 Informative 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 reference document (including any amendments) applies.

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

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™ 1363a-2004: "Standard Specifications For Public Key Cryptography - Amendment 1: Additional Techniques".

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

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

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

[i.5] Standards for Efficient Cryptography 4 (SEC 4): "Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV)".

[i.6] Antipa A., R. Gallant, and S. Vanstone: "Accelerated verification of ECDSA signatures", Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005: Springer, 2005, pp. 307-318.

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:

AES	Advanced Encryption Standard
CA	Certificate Authority
CAM	Cooperative Awareness Message
CRL	Certificate Revocation List
DENM	Decentralized Environmental Notification Message
DHAES	Diffie-Hellman: An Encryption Scheme
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ECQV	Elliptic Curve Qu-Vanstone

NOTE: Implicit Certificate Scheme.

G5	5,9 GHz radio communications
ITS	Intelligent Transport Systems
ITS-AID	ITS Application ID
ITS-S	Intelligent Transport Systems Station
LSB	Least Significant Bit
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)
TLS	Transport Layer Security
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 1609.2-2012 [i.2] (draft D12) and is described in table 1. The encoding of multi-byte elements of the presentation language shall always use network byte order, i.e. big endian byte order, if applicable.

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	`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 length encoding	The number in angle brackets gives the maximum number of octets. Depending on the maximum size, the first 1 byte, 2 bytes, 4 bytes 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 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. The maximum length shall be 2^{56} - 1, i.e. at most seven "1" bits followed by a "0" bit shall be used for the variable-length length encoding. The length of variable-length vectors with variable-length length encoding shall be encoded as positive integer using the minimum number of bits necessary	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

```
int_x    IntX;
```

This data type encodes an integer of variable length. The length of this integer is encoded by a number of 1 bits followed by a 0 bit, where the number of 1 bits is equal to the number of additional octets used to encode the integer besides those used (partially) to encode the length. The encoding of the length shall use at most 7 bits set to 1.

EXAMPLE: 00001010 encodes the integer 10, while 10001000 10001000 encodes the integer 2 184. The bits encoding the length of the element are coloured 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. `ecies_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. The `EccPoint` used in a `PublicKey` shall not have `EccPointType` `x_coordinate_only`.

- 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. The EccPoint used in a PublicKey shall not have EccPointType x_coordinate_only.

- 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 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. 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 as an unsigned integer. 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
ecies_nistp256	32

NOTE: Except inclusion of all remaining elements of the enumeration EccPointType that 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 point 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. The parameter Tlen according to NIST SP 800-38C [4] shall be set to Tlen = 128 (bits) and no associated data 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 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 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.

4.2.9 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. 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 x coordinate of the elliptic curve point resulting from multiplying the generator element by the ephemeral private key. The EccPointType of R shall be set to either compressed_lsb_y_0, compressed_lsb_y_1 or x_coordinate_only.

NOTE 1: 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.

NOTE 2: It is possible to add extra information by transferring the complete point R in a compressed form instead of only the x coordinate. This extra information may then be used for a faster signature verification algorithm as outlined in "Accelerated verification of ECDSA signatures" [i.6].

4.2.10 SignerInfo

```
struct {
    SignerInfoType type;
    select(type){
    case self:
                            ;
    case certificate_digest_with_sha256:
        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.
- certificate_digest_with_sha256: 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.

4.2.11 SignerInfoType

```
enum {
    self(0),
    certificate_digest_with_sha256(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.

4.2.12 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.

A canonical encoding for the EccPoint R contained in the signature field of a Certificate shall be used when calculating the SHA-256 hash from a Certificate. This canonical encoding shall temporarily replace the value of the EccPointType of the point R of the Certificate with x_coordinate_only for the hash computation.

NOTE 1: Except naming, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.2.6.

NOTE 2: The canonical encoding is used to remove the possibility of manipulating the certificate in a way that results in different HashedId8 identifiers for the same certificate by changing the EccPointType. Implementations that do not use the fast verification according to "Accelerated verification of ECDSA signatures" [i.6] cannot detect this manipulation.

4.2.13 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].

4.2.14 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, 01 January 2004.

NOTE 1: The period of 2^{32} seconds lasts about 136 years that is until 2140.

NOTE 2: This definition is identical to the one in IEEE 1609.2 Draft D17 [i.3], clause 6.3.31.

4.2.15 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, 01 January 2004.

NOTE: This definition is identical to the one in IEEE 1609.2 Draft D17 [i.3], clause 6.2.12.

4.2.16 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 254 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. Values greater than $1{,}134666^{254}$ nanoseconds are represented by the value 254, i.e. a day or longer. If the standard deviation is unknown, value 255 shall be used.

NOTE 1: This definition is identical to the one in IEEE 1609.2 Draft D17 [i.3], clause 6.2.11.

NOTE 2: This definition is currently unused in the security profiles in clause 7.

4.2.17 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.

NOTE 1: Except naming, this definition is identical to the one in IEEE 1609.2 Draft D12 [i.2], clause 6.3.5.

NOTE 2: This definition is currently unused in the security profiles in clause 7.

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

4.2.18 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.

4.2.19 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 decimetres. The value is interpreted as an asymmetric signed integer with an encoding as follows:

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

EXAMPLES: 0x0000 = 0 metre

0x03E8 = 100 metres

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

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

4.2.20 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.

4.2.21 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.

4.2.22 CircularRegion

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

This structure defines a circular region with `radius` given in metres and center at `center`. The region shall include all points on the reference ellipsoid's surface with a distance over surface of Earth equal to or less than the radius to the center point. For a location of type `ThreeDLocation`, i.e. the location contains an elevation component, the horizontal projection onto the reference ellipsoid is used to determine if the location lies within the circular region.

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

4.2.23 RectangularRegion

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

This structure defines a rectangular region by connecting the four points in the order (northwest.latitude, northwest.longitude), (northwest.longitude, southeast.longitude), (southeast.longitude, southeast.longitude), and (southeast.longitude, northwest.longitude). If two consecutive points P and Q define a line of constant latitude or longitude from P to Q, the left side of the line is defined as being outside of the polygon and the line itself and the right side of the line to be inside the rectangular region. A rectangular region is only valid if the location `northwest` is north of the location southeast. For a location of type `ThreeDLocation`, i.e. the location contains an elevation component, the horizontal projection onto the reference ellipsoid is used to determine if the location lies within the rectangular region.

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

4.2.24 PolygonalRegion

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

This variable-length vector describes a region by enumerating points on the region's boundary. If two consecutively specified points P and Q define a line of constant bearing from P to Q, the left side of the line is defined as being outside of the polygon and the line itself and the right side of the line to be inside the polygon. The points shall be linked to each other, with the last point linked to the first. No intersections shall occur and at least 3 and no more than 12 points shall be given. The specified region shall be continuous and shall not contain holes. For a location of type `ThreeDLocation`, i.e. the location contains an elevation component, the horizontal projection onto the reference ellipsoid is used to determine if the location lies within the polygonal region.

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

4.2.25 IdentifiedRegion

```
struct {
    RegionDictionary    region_dictionary;
    uint16              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. The details of `local_region` are unspecified.

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

4.2.26 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;
    HeaderField     header_fields<var>;
    Payload         payload_field;
    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 `2` shall be used. The `protocol_version` shall be increased, if the standard is changed in an incompatible way, i.e. the syntax is incompatible such that older implementations cannot parse the format or the semantic has been changed significantly.

- `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_field contains the message's payload. Multiple payloads in one message are not 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 1: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

NOTE 2: An example for a reason to increase the protocol_version is a change to the epoch in clause 4.2.15 and clause 4.2.16, which leads to incompatible messages. A counterexample would be an additional header field using the unknown case in clause 5.4. This header field can be ignored by old implementations, if the syntax is kept identical and the versions are compatible. Hence, the protocol_version should not be increased.

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 external data shall be included when calculating the signature, at the position where a non-external payload would be. In all other cases, the data shall be given as a variable-length vector containing opaque data.

NOTE 1: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

NOTE 2: Payloads of type signed_external are needed to add a signature in a non-intrusive way to an existing protocol stack, e.g. for extending an IPv6 stack.

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 its_aid:
        IntX                            its_aid;
    case signer_info:
        SignerInfo                      signer;
    case encryption_parameters:
        EncryptionParameters            enc_params;
    case recipient_info:
        RecipientInfo                   recipients<var>;
    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: a timestamp of type Time64, which shall describe the point in time, when the contents of the security headers are fixed prior to the signing process.

- generation_time_standard_deviation: a timestamp of type Time64WithStandardDeviation, which shall describe the point in time, when the contents of the security headers are fixed prior to the signing process. In addition to the timestamp, the confidence described by the standard deviation of the time value contained 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. The request shall be used to request pseudonym certificates and authorization authority certificates.

- its_aid: The ITS-AID of the application payload shall be given. The valid ITS-AIDs are specified according to ETSI TS 102 965 [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. Each recipient_info vector shall be preceeded by one encryption_parameters header field to determine the value of symm_key_len according to table 4.

For extensibility, the structure contains a variable field:

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

NOTE 1: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

NOTE 2: The generation_time_standard_deviation and the expiration header fields are currently unused in the security profiles in clause 7.

5.5 HeaderFieldType

```
enum {
    generation_time(0),
    generation_time_standard_deviation(1),
    expiration(2),
    generation_location(3),
    request_unrecognized_certificate(4),
    its_aid(5),
    signer_info(128),
    encryption_parameters(129),
    recipient_info(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_field field), including the encoding of their length. Also the length of the trailer_fields field and the type of the signature trailer field shall be included in the hash.

  If the payload_field field has type equal to signed_external, the data shall be included in the hash calculation immediately after the payload_field field, encoded as an opaque<var>, i.e. as if it was included.

  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.

- If the payload_field field type does not contain the keyword "signed" (unsecured or encrypted), then the trailer_fields field of the SecuredMessage shall not include a Signature.

- 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:
       EciesEncryptedKey             enc_key;
    unknown:
        opaque                       enc_key<var>;
    }
} RecipientInfo;
```

This structure contains information for the decryption of a message for a 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 EciesEncryptedKey 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 EciesEncryptedKey

```
 struct {
    extern SymmetricAlgorithm   symm_alg;
    extern uint32               symm_key_len;
    EccPoint                    v;
    opaque                      c[symm_key_len];
    opaque                      t[16];
} EciesEncryptedKey;
```

This structure defines how to transmit an ECIES-encrypted symmetric key as defined in IEEE Std 1363a-2004 [i.1]. The EccPoint v contains the sender's ECC ephemeral key used for the Elliptic Curve Encryption Scheme. This ephemeral key v shall only be used once and for every encryption a new key shall be generated. The vector c contains the encrypted (AES) key. The vector t contains the authentication tag. The symm_key_len defines the length of vector c containing the encrypted (AES) key and 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 EciesEncryptedKey structure shall be preceded by an according EncryptionParameters structure.

Further parameters used for the encryption and decryption using ECIES shall be:

- The parameters P_1 and P_2 shall be empty strings.
- ECSVDP-DHC shall be used as secret value derivation primitive.
- The stream cipher used shall be based on KDF2 using SHA-256.
- As MAC, MAC1 shall be used with SHA-256 and tBits = 128.
- The length of the key (input) to MAC1 shall be 256 bits.
- The encryption shall use non-DHAES mode.
- Octet strings shall be interpreted using LSB compressed representation or uncompressed representation for the ECC points.

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

SymmetricAlgorithm 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;
    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 `2` for conformance with the present document. The `version` shall be increased, if the standard is changed in an incompatible way, i.e. the syntax is incompatible such that older implementations cannot parse the format or the semantic has been changed significantly.

- Information on this certificate's signer is given `signer_info field`. The signer_info shall be of type `self`, `certificate_digest_with_sha256`, `certificate_digest_with_other_algorithm`, or `reserved`.

- `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. Each certificate shall include at least one `validity_restriction` of type `time_end`, `time_start_and_end`, or `time_start_and_duration`.

- `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. The `reconstruction_value` may be used for implicit certificates using ECQV [i.5].

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 Enrolment 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`.
- Enrolment CAs, which sign enrolment 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>;
    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, which may be used for implicit certificates using ECQV [i.5].
- `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.
- `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),
    (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. Bit 7 to bit 5 denote the ITS-S's assurance levels, bit 4 to bit 2 are reserved for future use and bit 1 and bit 0 denote the confidence.

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

NOTE 1: This definition is not available in IEEE 1609.2 Draft D12 [i.2].

NOTE 2: A process should be defined how to evaluate each implementation and how to assign a corresponding subject assurance according to the evaluation result(s). However, this process is out of scope of the present document.

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`, or `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.

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.

These HeaderField elements shall be included in all CAMs. 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 exactly one field of the types certificate_digest_with_sha256, certificate_chain or certificate, according to the following rules:
  - In the normal case, the signer_info field of type certificate_digest_with_sha256 shall be included.
  - Instead of including a field of type certificate_digest_with_sha256, 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 certificate, it shall include a field of type certificate immediately in its next CAM, instead of including a field of type certificate_digest_with_sha256. 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 authorization ticket and not of the authorization authority 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_sha256. If the ITS-S finds a HashedId3 of its own, currently used authorization authority in that list, it shall include a signer_info field of type certificate_chain containing the currently used authorization ticket and authorization authority certificate immediately in its next CAM, instead of including a signer_info field of type certificate_digest_with_sha256.
- generation_time: this field shall contain the current absolute time. The generation_time is valid, if it is in the validity period of the certificate referenced by the signer_info.
- its_aid: this field shall encode the ITS-AID for CAMs according to ETSI TS 102 965 [7].

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_sha256 instead of a signer_info HeaderField 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 1: HashedId3 elements can be formed by using the least significant three bytes of the corresponding HashedId8 .

None of the possible HeaderField cases shall be included more than once. All other HeaderField types defined in clause 5 shall not be used. Future HeaderField types may be included. Any other HeaderField types included shall not be used to determine the validity of the message.

A Payload element shall be included for all CAMs. This element shall be of type signed and contain the CAM payload.

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 complete payload_field field.
    - The length of the variable-length vector trailer_fields and the type of the signature trailer field.

CAMs shall not be encrypted.

NOTE 2: Table 6 illustrates which parts of a SecuredMessage are taken into account when generating the signature of a message.

Table 6: Example for the ECDSA signature generation for a SecuredMessage

Element	Description
SecuredMessage	
uint8 protocol_version	Covered by the signature
HeaderField header_fields<var>	
…	
Payload payload_fields<var>	
…	
TrailerField trailer_fields<var>	
TrailerFieldType type	
PublicKeyAlgorithm algorithm	Not covered by the signature
EcdsaSignature ecdsa_signature	
EccPoint R	
EccPointType type	
opaque x[32]	ECDSA signature (r,s)
opaque s[32]	

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.

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. The generation_time is valid, if it is in the validity period of the certificate referenced by the signer_info.
- generation_location: this field shall contain the current location of the ITS-S at the point in time the contents of the security headers are fixed prior to the signing process. The generation_location is valid, either if there is no geographic validity restriction in the certificate referenced by the signer_info, or if it is inside the geographic validity restriction of this certificate.
- its_aid: this field shall encode the ITS-AID for DENMs according to ETSI TS 102 965 [7].

None of the possible HeaderField cases shall be included more than once. All other HeaderField types defined in clause 5 shall not be used. Future HeaderField types may be included. Any other HeaderField types included shall not be used to determine the validity of the message.

A Payload element shall be included for all DENMs. This element shall be of type signed and contain the DENM payload.

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 complete payload_field field.
    - The length of the variable-length vector trailer_fields and the type of the signature trailer 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.

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. The generation_time is valid, if it is in the validity period of the certificate referenced by the signer_info.
- generation_location: this field shall contain the current location of the ITS-S at the point in time the contents of the security headers are fixed prior to the signing process. The generation_location is valid, either if there is no geographic validity restriction in the certificate referenced by the signer_info, or if it is inside the geographic validity restriction of this certificate.
- its_aid: this field shall encode an ITS-AID according to ETSI TS 102 965 [7]. This field shall not encode an ITS-AID that is reserved for use with other security profiles. The present document covers the ITS-AIDs for CAM and DENM.

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

A 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 complete payload_field field. 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 the type of the signature trailer field.

7.4 Profiles for certificates

7.4.1 Introduction

Clause 7.4 defines which types of variable fields shall always be included in certificates.

The version field of a certificate shall be set according to clause 6.1.

The following SubjectAttribute elements shall be included:

- verification_key: this field shall contain the public key of the key pair that is used to sign and verify message or certificate signatures.
- assurance_level: this field shall contain the assurance level of the sender or certificate authority. A certificate shall contain an assurance level that is equal to or lower than the assurance level of the certificate referenced by the signer_info. If the assurance level is unknown for the certificate then the default assurance level 0 shall be used.

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. If the signer_info is different from self, then the validity period defined by time_end, time_start_and_end or time_start_and duration shall be within the validity period of the certificate referenced by the signer_info.

A certificate shall contain a validity restriction of type region, if the certificate referenced by the signer_info contains a validity restriction of type region. Every certificate with a validity restriction of type region shall contain a region that is covered by the certificate referenced by the signer_info. For the field signer_info, exactly one of the following types shall be included:

- certificate_digest_with_sha256
- certificate_digest_with_other_algorithm
- self

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

All certificates shall contain a Signature field containing a signature calculated over these fields of the Certificate data structure:

- The version
- The signer_info
- The subject_info
- The subject_attributes vector including its length
- The validity_restrictions vector including its length

Every certificate containing an its_aid_list or its_aid_ssp_list subject attribute shall contain a subset of the permissions that are contained in the certificate referenced by the signer_info. An its_aid in an its_aid_list shall be interpreted as containing a superset of all possible service specific permissions of this its_aid.

7.4.2 Authorization tickets (pseudonymous certificates)

This clause defines additional aspects of authorization tickets (i.e. pseudonymous certificates) as defined in ETSI TS 102 940 [6].

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

- `certificate_digest_with_sha256`.

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 encoded as `0x00` (empty name field).

These `SubjectAttribute` elements shall be included in addition to those specified in clause 7.4.1 for all certificates:

- `its_aid_ssp_list`: this field shall contain a list of ITS-AIDs with associated Service Specific Permissions (SSP). For each ITS-AID only one `ItsAidSsp` shall be used.

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

7.4.3 Enrolment credential (long-term certificates)

This clause defines additional aspects of enrolment credentials (i.e. long-term certificates) as defined in ETSI TS 102 940 [6].

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

- `certificate_digest_with_sha256`.

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

These `SubjectAttribute` elements shall be included in addition to those specified in clause 7.4.1 for all certificates:

- `its_aid_ssp_list`: this field shall contain a list of ITS-AIDs with associated Service Specific Permissions (SSP). For each ITS-AID only one `ItsAidSsp` shall be used.

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

NOTE: The `its_aid_ssp_list` is used for enrolment credentials to enforce that an ITS-S cannot expand its own service specific permissions in authorization tickets through manipulation of requests to the CA.

7.4.4 Certificate authority certificates

This clause defines additional aspects of certificate authority certificates.

The following `SignerInfo` fields shall be included:

- For root certificate authority certificates, the `signer_info` field shall be set to `self`.
- For other certificate authorities, the `signer_info` field shall be set to `certificate_digest_with_sha256`.

In the `SubjectInfo` field of the CA certificate, `subject_type` shall be set to one of these types:

- `authorization_authority`, for authorization authorities, i.e. certificate authorities issuing authorization tickets.
- `enrollment_authority`, for enrolment authorities, i.e. certificate authorities issuing enrolment credentials.

- root_ca, for root certificate authorities.

These SubjectAttribute elements shall be included in addition to those specified in clause 7.4.1 for authorization authority and enrolment authority certificates:

- its_aid_list: this field shall contain a list of ITS-AIDs. Each ITS-AID shall be unique in the its_aid_list.

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

NOTE: The authorization and enrolment authority certificates contain an its_aid_list, because a CA should not be able to create certificates for ITS stations containing ITS-AIDs that the CA was not authorized to by the root CA.

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 NIST P-256 based signature is attached.

Table A.1: An example signed header for CAM

Element	Value	Description	Length in octets
SecuredMessage			
uint8 protocol_version	0x02		1
HeaderField header_fields<var>	0x15	length: 21 octets	1
HeaderFieldType type	0x80	signer_info	1
SignerInfoType signer_info	0x01	certificate_digest_with_sha256	1
HashedId8 digest	[…]		8
HeaderFieldType type	0x00	generation_time	1
Time64 generation_time	[…]		8
HeaderFieldType type	0x05	its_aid	1
IntX its_aid	[…]	ITS-AID for CAM	1
Payload payload_field		payload	
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	0x00	x_coordinate_only	1
opaque x[32]	[…]		32
opaque s[32]	[…]		32
The total size of the security header structure is 93 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	0x02		1
SignerInfo signer_info			
SignerInfoType type	0x01	certificate_digest_with_sha256	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	0x02	compressed_lsb_y_0	1
opaque x[32]	[…]		32
SubjectAttributeType type	0x02	assurance_level	1
SubjectAssurance assurance_level	0x83	level_4_confidence_3	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	0x00	x_coordinate_only	1
opaque x[32]	[…]		32
opaque s[32]	[…]		32
The total size of this certificate is 132 octets.			

Annex B (informative): Usage of ITS-AID and SSPs

An incoming secured message should only be accepted by the receiver if the payload of the secured message is consistent with the ITS-AID and SSP in the certificate. This consistency should be checked in two ways:

1) Within the security processing, the ITS-AID in the certificate can be checked for consistency with the `its_aid` field in the SecuredMessage format.

2) At the point at which the data is processed (which may be in the receiving facilities layer or in the receiving application layer), the data can be checked for consistency with the ITS-AID and the SSP from the certificate. Architecturally, this check is carried out by the processing entity that processes the data payload of the SecuredMessage, not by the security processing services. This is because the security processing services cannot and should not be expected to be able to parse the data of all possible different applications and facilities. Thus, a full definition of a data exchange for applications or facilities that use signed messages should include a specification of the ITS-AID, a specification of the SSP, and a definition of what it means for the data itself to be consistent with the ITS-AID and SSP.

The use of ITS-AID and SSP therefore includes the following steps:

1) At the design stage, the group defining a given data exchange determines whether the exchanges are to be signed with ETSI TS 103 097 certificates. If they are, the group reserves an ITS-AID and defines an SSP.

2) When an ITS-Station is initialized with the ability to carry out a data exchange, it requests certificates with the appropriate ITS-AID and SSP.

3) An Authorization Authority determines whether the ITS-Station is entitled to that ITS-AID and SSP, using methods outside the scope of the present document to make that determination. It issues the certificates to the ITS-S.

4) The sending ITS-Station generates a message that is consistent with the ITS-AID and SSP, and uses the private key corresponding to the certificate to sign that message.

5) On the receiving side, the security processing checks that the message was correctly cryptographically signed, is not a replay (if appropriate), etc.

6) On the receiving side, the data processing entity (which may be an application or the facilities layer) uses the ITS-AID and SSP from the certificate to check that the data is consistent with those permissions. This means that the ITS-AID and SSP should be made available to the data processing entity, for example by passing them across an interface from the security processing, or by passing the entire certificate and letting the data processing entity extract the ITS-AID and SSP from the certificate, or by some other means.

NOTE 1: The ITS-AID and SSP are contained in the certificate, which is cryptographically authenticated and authorized by the Authorization Authority. Because of this cryptographic authentication, it is impossible for the certificate holder to change their permissions without causing cryptographic authentication to fail.

NOTE 2: The ETSI TS 103 097 certificate format allows a certificate to contain multiple (ITS-AID, SSP) pairs. In this case, the receiving side processing is expected to know which ITS-AID is to be used in conjunction with an incoming message.

One way to make the concept of SSP future proof is to add the version number of the corresponding facility to the SSP.

The interpretation of the SSP is specific for each facility. One possible way to implement it is to use a bit map to define which permissions a sender is authorized to use.

The bit value "1" then means that the sender is authorized to use the corresponding feature and consequently the bit value "0" means that the sender is not authorized to use it.

History

Document history		
V1.1.1	April 2013	Publication
V1.2.1	June 2015	Publication