ETSI TS 103 097 V1.3.1 (2017-10)

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

Reference
RTS/ITS-00540

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

Contents

Intellectual Property Rights

Essential patents

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 (https://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.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

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 policies require that data structures such as messages used in Intelligent Transport Systems are secured when stored or transferred. For interoperability reasons, a common format for secure data structures featuring security headers and public key certificates needs to be provided.

The present document provides these definitions as a profile of the base standard IEEE Std 1609.2™-2016 and its amendment IEEE 1609.2a™-2017 [1]. A profile makes use of the definitions in the base standard and defines the use of particular subsets or options available in the base standard. This implies that the present document is to be read and interpreted together with that base standard.

The present document contains material from IEEE Std 1609.2-2016 [1] and its amendment(s), reprinted with permission from IEEE, and Copyright © 2016.

1 Scope

The present document specifies the secure data structure including header and certificate formats for Intelligent Transport Systems.

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 referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://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 Std 1609.2™-2016: "IEEE Standard for Wireless Access in Vehicular Environments -- Security Services for Applications and Management Messages", as amended by IEEE Std 1609.2a™-2017: "Standard for Wireless Access In Vehicular Environments -- Security Services for Applications and Management Messages Amendment 1".

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

[3] Recommendation ITU-T X.696 (08/2014): "Information Technology-Specification of Octet Encoding Rules (OER)".

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 referenced 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] ETSI TS 102 940: "Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management".

[i.2] ETSI TS 102 941: "Intelligent Transport Systems (ITS); Security; Trust and Privacy Management".

3 Abbreviations

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

AA Authorization Authority
ASN.1 Abstract Syntax Notation One
AT Authorization Ticket

CA	Certification Authority
CAM	Cooperative Awareness Message
COER	Canonical Octet Encoding Rules
CRL	Certificate Revocation List
CTL	Certificate Trust List
DENM	Decentralized Environmental Notification Message
EA	Enrolment Authority
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ITS	Intelligent Transport Systems
ITS-AID	ITS Application ID
ITS-S	Intelligent Transport Systems Station
SSP	Service Specific Permissions
TLM	Trust List Manager

4 Basic format elements

Data structures in the present document are defined using Abstract Syntax Notation 1 (ASN.1) and shall be encoded using the Canonical Octet Encoding Rules (COER) as defined in Recommendation ITU-T X.696 [3]. This includes some data structures in the present document for which a "canonical encoding" is used as defined in IEEE Std 1609.2 [1].

Clause 5 and 6 specify and describe the data structures with reference to IEEE Std 1609.2 [1]. The corresponding ASN.1 module is defined in annex A.

The validity of a certificate shall be assessed as defined in IEEE Std 1609.2 [1] clause 5.1, using the Hash ID-based revocation method for EA and AA certificates, and no revocation method for authorization tickets and enrolment credentials.

NOTE 1: The CRL for EA and AA certificates is defined in ETSI TS 102 941 [i.2].

NOTE 2: The rules for verification of the Root CA certificate against the CTL are defined in ETSI TS 102 941 [i.2].

The validity of signed data shall be assessed as defined in IEEE Std 1609.2 [1] clause 5.2.

5 Specification of secure data structure

5.1 EtsiTs103097Data

A secure data structure shall be of type `EtsiTs103097Data` as defined in annex A, which corresponds to a `Ieee1609Dot2Data` as defined in IEEE Std 1609.2 [1] clause 6.3.2, with the constraints defined in this clause, in clause 5.2 and in clause 5.3.

The type `Ieee1609Dot2Data` shall support the following options in the component content:

- The option `unsecuredData` shall be used to encapsulate an unsecured data structure.
- The option `signedData`, corresponding to the type `SignedData` as defined in IEEE Std 1609.2 [1] clause 6.3.4, shall be used to transfer a data structure with a signature.
- The option `encryptedData`, corresponding to the type `EncryptedData` as defined in IEEE Std 1609.2 [1] clause 6.3.30, shall be used to transfer an encrypted data structure.

The following corresponding profiles of the type `EtsiTs103097Data` are defined in annex A:

- The parameterized type `EtsiTs103097Data-Signed` using the Ieee1609Dot2Data option `signedData` containing the data structure in the component tbdData.payload.data.

- The parameterized type `EtsiTs103097Data-SignedExternalPayload` using the `Ieee1609Dot2Data` option `signedData` containing the digest of the data structure in the component `tbdData.payload.extDataHash`.
- The parameterized type `EtsiTs103097Data-Encrypted`, using the `Ieee1609Dot2Data` option `encryptedData` containing the encrypted data structure in the component `ciphertext.aes128ccm.ccmCiphertext`.
- The parameterized type `EtsiTs103097Data- SignedAndEncrypted`, using the `Ieee1609Dot2Data` option `EncryptedData`, containing an encrypted `EtsiTs103097Data-Signed`.

5.2 SignedData

The type `SignedData` shall have the following constraints:

The component `hashId` of `SignedData` shall indicate the hash algorithm to be used to generate the hash of the message according to IEEE Std 1609.2 [1] clauses 6.3.5 and 5.3.3.

The component `tbsData` of `SignedData` shall be of type `ToBeSignedData` as defined in IEEE Std 1609.2 [1] clause 6.3.6. The type `ToBeSignedData` shall have the component `payload` of type `SignedDataPayload` as defined in IEEE Std 1609.2 [1] clause 6.3.7, containing either:

- the component `data`, containing the payload to be signed as an `Ieee1609Dot2Data`, or
- the component `extDataHash`, containing the hash of data that is not explicitly transported within the structure.

The type `ToBeSignedData` shall have the component `headerInfo` of type `HeaderInfo` as defined in IEEE Std 1609.2 [1] clause 6.3.9, and constrained to have the following security headers:

- The component `psid` containing the ITS-AID corresponding to the contained message.
- The component `generationTime` as defined in IEEE Std 1609.2 [1], always present.
- The component `expiryTime`, as defined in IEEE Std 1609.2 [1], present or absent according to the specification of message profiles in clause 7.
- The component `generationLocation`, as defined in IEEE Std 1609.2 [1], present or absent according to the specification of message profiles in clause 7.
- The component `p2pcdLearningRequest` always absent.
- The component `missingCrlIdentifier` always absent.
- The component `encryptionKey`, as defined in IEEE Std 1609.2 [1], present or absent according to the specification of message profiles in clause 7.
- The component `inlineP2pcdRequest`, as defined in IEEE Std 1609.2 [1], present or absent according to the specification of message profiles in clause 7.
- The component `requestedCertificate`, as defined in IEEE Std 1609.2 [1], present or absent according to the specification of message profiles in clause 7.

The component `signer` of `SignedData` shall be of type `SignerIdentifier` as defined in IEEE Std 1609.2 [1] clause 6.3.24, and constrained to one of the following choices:

- `digest`, containing the digest of the signing certificate as defined in IEEE Std 1609.2 [1] clause 6.3.26.
- `certificate`, constrained to only one entry in the `SequenceOfCertificate` list of type `TS103097Certificate`, containing the signing certificate as defined in clause 6 of the present document.

The component `signature` of `SignedData` shall be of type `Signature` as defined in IEEE Std 1609.2 [1] clause 6.3.28 and shall contain the ECDSA signature as defined in IEEE Std 1609.2 [1] clauses 6.3.29, 6.3.29a and 5.3.1.

5.3 EncryptedData

The type `EncryptedData` shall have the following constraints:

The component `recipients` of `EncryptedData` shall be of type `SequenceOfRecipientInfo` as defined in IEEE Std 1609.2 [1] clause 6.3.31. Every entry shall be either of option `pskRecipInfo` as defined in IEEE Std 1609.2 [1] clause 6.3.32, of option `certRecipInfo`, or of option `signedDataRecipInfo`, as defined in IEEE Std 1609.2 [1] clause 6.3.34.

The encryption scheme used shall be ECIES as defined in IEEE Std 1609.2 [1] clause 5.3.5. The component `ciphertext` of `EncryptedData` shall be of type `SymmetricCiphertext` as defined in IEEE Std 1609.2 [1] clause 6.3.37 and contain a `EtsiTs103097Data` encrypted according to IEEE Std 1609.2 [1] clauses 6.3.38 and 5.3.8.

6 Specification of certificate format

A certificate contained in a secure data structure shall be of type `EtsiTs103097Certificate` as defined in annex A, which corresponds to a single `ExplicitCertificate` as defined in IEEE Std 1609.2 [1] clause 6.4.6, with the constraints defined in this clause.

The component `toBeSigned` of the type `EtsiTs103097Certificate` shall be of type `ToBeSignedCertificate` as defined in IEEE Std 1609.2 [1] clause 6.4.8 and constrained as follows:

- The component `id` of type `CertificateId` constrained to choice type `name` or `none`.
- The component `cracaId` set to 000000'H.
- The component `crlSeries` set to 0'D.
- The component `validityPeriod` with no further constraints.
- The component `region` of type `GeographicRegion` as defined in IEEE Std 1609.2 [1], present or absent according to the specification of certificate profiles in clause 7.
- The component `assuranceLevel` of type `SubjectAssurance`, as defined in IEEE Std 1609.2 [1], present or absent according to the specification of certificate profiles in clause 7.
- The component `appPermissions` of type `SequenceOfPsidSsp` as defined in IEEE Std 1609.2 [1], present or absent according to the specification of certificate profiles in clause 7.
- The component `certIssuePermissions` of type `SequenceOfPsidGroupPermissions`, as defined in IEEE Std 1609.2 [1], present or absent according to the specification of certificate profiles in clause 7.
- At least one of the components `appPermissions` and `certIssuePermissions` shall be present.
- The component `certRequestPermissions` absent.
- The component `canRequestRollover` absent.
- The component `encryptionKey` of type `PublicEncryptionKey` as defined in IEEE Std 1609.2 [1], present or absent according to the specification of certificate profiles in clause 7.
- The component `verifyKeyIndicator` of type `VerificationKeyIndicator` as defined in IEEE Std 1609.2 [1], present and constrained to the choice verificationKey.

The component `signature` of `EtsiTs103097Certificate` shall be of type `Signature` as defined in IEEE Std 1609.2 [1] clause 6.3.28 and shall contain the signature, calculated by the signer identified in the issuer component, as defined in IEEE Std 1609.2 [1] clauses 6.3.29, 6.3.29a and 5.3.1.

7 Security profiles

7.1 Profiles for messages

7.1.1 Security profile for CAMs

The secure data structure containing Cooperative Awareness Messages (CAMs) shall be of type `EtsiTs103097Data-Signed` as defined in clause 5.1 and annex A, containing the CAM as the `ToBeSignedDataContent`, with the additional constraints defined in clause 5.2 and this clause:

- The component `signer` of `SignedData` shall be constrained as follows:
  - As default, the choice `digest` shall be included.
  - The choice `certificate` shall be included once, one second after the last inclusion of the choice `certificate`.
  - If the ITS-S receives a CAM signed by a previously unknown AT, it shall include the choice `certificate` immediately in its next CAM, instead of including the choice `digest`. In this case, the timer for the next inclusion of the choice `certificate` shall be restarted.
  - If an ITS-S receives a CAM that includes a `tbsdata.headerInfo` component of type `inlineP2pcdRequest`, then the ITS-S shall evaluate the list of certificate digests included in that component: If the ITS-S finds a certificate digest of the currently used authorization ticket in that list, it shall include a the choice `certificate` immediately in its next CAM, instead of including the choice `digest`.
- The component `tbsdata.headerInfo` of `SignedData` shall be further constrained as follows:
  - `psid`: this component shall encode the ITS-AID value for CAMs as assigned in ETSI TS 102 965 [2].
  - The component `inlineP2pcdRequest` shall be included and shall contain the digests of certificates currently unknown to the ITS-Station in the following cases:
    - if the ITS-S received a CAM with the component `signer` of `SignedData` set to the choice `digest`, and this digest points to an unknown authorization ticket;
    - if the ITS-S received a message with the component `signer` of `SignedData` set to the choice `certificate`, and this certificate is signed by an unknown authorization authority certificate, i.e. includes the component `issuer` referencing an unknown certificate.
  - `requestedCertificate`: If an ITS-S receives a CAM with the component `tbsdata.headerInfo` including a the component `inlineP2pcdRequest`, then the ITS-S shall evaluate the list of digests included in that component: If the ITS-S finds a digest of a valid certification authority certificate, it shall include the component `requestedCertificate` containing the requested certificate immediately in its next CAM:
    - unless before the generation of the next CAM, the ITS-S received another CAM including the component `requestedCertificate` containing the requested certification authority certificate: in this case the request shall be discarded;
    - unless the component `signer` of `SignedData` is of choice `certificate` according to the rules defined above: in this case the request shall be kept pending and the certificate shall be inserted in the next possible CAM, according to the same conditions.

- All other components of the component `tbsdata.headerInfo` allowed to be present according to clause 5 shall not be used and be absent.

7.1.2 Security profile for DENMs

The secure data structure containing Decentralized Environmental Notification Messages (DENMs) shall be of type `EtsiTs103097Data-Signed` as defined in clause 5.1 and annex A, containing the DENM as the `ToBeSignedDataContent`, with the additional constraints defined clause 5.2 and in this clause:

- The component `signer` of `SignedData` shall be of choice `certificate`.
- The component `tbsdata.headerInfo` of SignedData shall be further constrained as follows:
  - `generationLocation`: shall be present.
  - `psid`: this component shall encode the ITS-AID value for DENMs as assigned in ETSI TS 102 965 [2].
- All other components of the component `tbsdata.headerInfo` allowed to present according to clause 5 shall not be used and be absent.

7.1.3 Generic security profile for other signed messages

The secure data structure containing signed messages other than CAM and DENM shall be of type:

- `EtsiTs103097Data-Signed` as defined in clause 5.1 and annex A, containing the message as the `ToBeSignedDataContent`, or of type;
- `EtsiTs103097Data-SignedExternalPayload` as defined clause 5.1 and in annex A, containing the message digest;

with the additional constraints defined in clause 5.2.

7.1.4 Security profile for encrypted messages

The secure data structure containing encrypted messages shall be of type `EtsiTs103097Data-Encrypted` as defined in clause 5.1 and annex A, containing the message as the `ToBeEncryptedDataContent`, with the additional constraints defined in clause 5.3.

7.1.5 Security profile for signed and encrypted messages

The secure data structure containing signed and then encrypted messages shall be of type `EtsiTs103097Data-SignedAndEncrypted` as defined in clause 5.1 and annex A, containing the message as the `ToBesignedAndEncryptedDataContent`. This corresponds to a `EtsiTs103097Data` of type `EtsiTs103097Data-Encrypted`, containing a EtsiTs103097Data of type `EtsiTs103097Data-Signed`, containing the message as the `ToBeSignedDataContent`.

7.2 Profiles for certificates

7.2.1 Authorization tickets

This clause defines additional aspects of authorization tickets as defined in ETSI TS 102 940 [i.1]. Authorization tickets shall be of type `EtsiTs103097Certificate` as defined in clause 6, with the following constraints:

The component `issuer` shall be of choice `sha256AndDigest` or `sha384AndDigest` as defined in IEEE Std 1609.2 [1] clause 6.4.7.

The `toBeSigned` component `appPermissions` shall be used to indicate message signing permissions, i.e. permissions to sign a `EtsiTs103097Data.`

The `toBeSigned` component `CertificateId` shall be set to the choice `none`.

The `toBeSigned` component `certIssuePermissions` shall be absent.

7.2.2 Enrolment credential

This clause defines additional aspects of enrolment credentials (i.e. long-term certificates) as defined in ETSI TS 102 940 [i.1]. Enrolment credentials shall be of type `EtsiTs103097Certificate` as defined in clause 6, with the following constraints:

The component `issuer` shall be of choice `sha256AndDigest` or `sha384AndDigest` as defined in IEEE Std 1609.2 [1] clause 6.4.7.

The `toBeSigned` components `appPermissions` shall be used to indicate message signing permissions, i.e. permissions to sign a certificate request message contained in a `EtsiTs103097Data`.

NOTE: An example of certificate request messages is given in ETSI TS 102 941 [i.2].

The `toBeSigned` component `CertificateId` shall be set to the choice `name` and shall contain a unique name associated to the enrolment credential.

The `toBeSigned` component `certIssuePermissions` shall be absent.

7.2.3 Root CA certificates

This clause defines additional aspects of Root CA certificates as defined in ETSI TS 102 940 [i.1]. Root CA certificates shall be of type `EtsiTs103097Certificate` as defined in clause 6, with the following constraints:

The component `issuer` shall be set as follows:

- For root certification authority certificates, the component `issuer` shall be set to `self`.
- For root certification authority link certificates, the component `issuer` shall be set to `sha256AndDigest` or `sha384AndDigest` as defined in IEEE Std 1609.2 [1] clause 6.4.7.

These `toBeSigned` components shall be included in addition to those specified in clause 6:

- `certIssuePermissions` shall be used to indicate issuing permissions, i.e. permissions to sign subordinate certification authority certificates with certain permissions.
- `appPermissions` shall be used to indicate permissions to sign:
  - CRLs and contain the ITS-AID for the CRL as assigned in ETSI TS 102 965 [2].
  - CTLs and contain the ITS-AID for the CTL as assigned in ETSI TS 102 965 [2].

The `toBeSigned` component `CertificateId` shall be set to the choice `name` and shall contain a unique name associated to the root certification authority.

7.2.4 Subordinate certification authority certificates

This clause defines additional aspects of subordinate certification authority certificates, i.e. enrolment and authorization authorities certificates as defined in ETSI TS 102 940 [i.1]. Subordinate certification authority certificates shall be of type `EtsiTs103097Certificate` as defined in clause 6, with the following constraints:

The component `issuer` shall be set to `sha256AndDigest` or `sha384AndDigest` as defined in IEEE Std 1609.2 [1] clause 6.4.7.

These `toBeSigned` components shall be included in addition to those specified in clause 6:

- `encryption_key`: this component shall contain a public encryption key for ITS-Stations to encrypt messages to the enrolment / authorization authority.

- `certIssuePermissions`: this component shall be used to indicate issuing permissions, i.e. permissions to sign an enrolment credential / authorization ticket with certain permissions.
- `appPermissions`: this component shall be used to indicate message signing permissions, i.e. permissions to sign certificate response messages contained in a `EtsiTs103097Data`.

NOTE: An example of certificate response messages is given in ETSI TS 102 941 [i.2].

The `toBeSigned` component `CertificateId` shall be set to the choice `name` contain a unique name associated to the certification authority, or shall be set to the choice `none`.

7.2.5 Trust List Manager certificate

This clause defines additional aspects of Trust List Manager certificates. Trust List Manager certificates shall be of type `EtsiTs103097Certificate` as defined in clause 6, with the following constraints:

The component `issuer` shall be set as follows:

- For Trust List Manager certificates, the component `issuer` shall be set to `self`.
- For Trust List Manager link certificates, the component `issuer` shall be set to `sha256AndDigest` or `sha384AndDigest` as defined in IEEE Std 1609.2 [1] clause 6.4.7.

These `toBeSigned` components shall be included in addition to those specified in clause 6:

- `region`: this component shall contain the geographic validity restriction associated to the Trust List Manager.
- `appPermissions`: this component shall contain the ITS-AID for the CTL as assigned in ETSI TS 102 965 [2].

The `toBeSigned` component `CertificateId` shall be set to the choice `name` and contain the unique name string associated to the TLM.

These `toBeSigned` components shall be absent:

- `encryptionKey`.
- `certIssuePermissions`.

Annex A (normative): ASN.1 Modules

A.1 ETSI TS 103 097 ASN.1 Module

This clause defines the normative ASN.1 module for the present document. The ASN.1 modules imports data types from the ASN.1 modules defined in IEEE Std 1609.2 [1].

```
EtsiTs103097Module
{ itu-t(0) identified-organization(4) etsi(0) itsDomain(5) wg5(5) ts(103097) v1(0) }

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

IMPORTS

Ieee1609Dot2Data, ExplicitCertificate

FROM

IEEE1609dot2 {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) base (1) schema (1) major-version-2(2)};

EtsiTs103097Certificate::= ExplicitCertificate
        (WITH COMPONENTS{...,
            toBeSigned (WITH COMPONENTS{...,
              id (WITH COMPONENTS{...,
                linkageData ABSENT,
                binaryId ABSENT
              }),
              certRequestPermissions ABSENT,
              canRequestRollover ABSENT
            })
         })

SingleEtsiTs103097Certificate ::= SEQUENCE {
  only  EtsiTs103097Certificate
}


EtsiTs103097Data::=Ieee1609Dot2Data (WITH COMPONENTS {...,
  content (WITH COMPONENTS {...,
    signedData (WITH COMPONENTS {..., -- constraints on signed data headers
      tbsData (WITH COMPONENTS {
        headerInfo (WITH COMPONENTS {...,
          generationTime PRESENT,
          p2pcdLearningRequest ABSENT,
          missingCrlIdentifier ABSENT
        })
      }),
      signer (WITH COMPONENTS {...,  --constraints on the certificate
        certificate (WITH COMPONENT (SingleEtsiTs103097Certificate))
      })
    }),
    encryptedData (WITH COMPONENTS {..., -- constraints on encrypted data headers
      recipients  (WITH COMPONENT (
       (WITH COMPONENTS {...,
         symmRecipInfo ABSENT,
         rekRecipInfo ABSENT
        })
      ))
    }),
    signedCertificateRequest ABSENT
  })
})

EtsiTs103097Data-Signed {ToBeSignedDataContent} ::= EtsiTs103097Data (WITH COMPONENTS {...,
  content (WITH COMPONENTS {
    signedData (WITH COMPONENTS {...,
      tbsData (WITH COMPONENTS {
        payload (WITH COMPONENTS {
          data (WITH COMPONENTS {...,
```

```
            content (WITH COMPONENTS {
               unsecuredData (CONTAINING ToBeSignedDataContent)
            })
          }) PRESENT
        })
      })
    })
  })
})

EtsiTs103097Data-SignedExternalPayload ::= EtsiTs103097Data (WITH COMPONENTS {...,
  content (WITH COMPONENTS {
    signedData (WITH COMPONENTS {...,
      tbsData (WITH COMPONENTS {
        payload (WITH COMPONENTS {
          extDataHash (WITH COMPONENTS {
             sha256HashedData PRESENT
          }) PRESENT
        })
      })
    })
  })
})

EtsiTs103097Data-Encrypted {ToBeEncryptedDataContent} ::= EtsiTs103097Data (WITH COMPONENTS {...,
  content (WITH COMPONENTS {
    encryptedData (WITH COMPONENTS {...,
      ciphertext (WITH COMPONENTS {...,
        aes128ccm (WITH COMPONENTS {...,
          ccmCiphertext (CONSTRAINED BY {-- ccm encryption of -- ToBeEncryptedDataContent})
        })
      })
    })
  })
})


EtsiTs103097Data-SignedAndEncrypted {ToBesignedAndEncryptedDataContent} ::= EtsiTs103097Data-
Encrypted {EtsiTs103097Data-Signed {ToBesignedAndEncryptedDataContent}}

END
```

A.2 IEEE 1609.2 ASN.1 modules

This clause provides the relevant ASN.1 modules from IEEE Std 1609.2 [1] (and its amendments), reprinted with permission from IEEE, Copyright © 2016.

```
IEEE1609dot2 {iso(1) identified-organization(3) ieee(111)
standards-association-numbered-series-standards(2) wave-stds(1609)
dot2(2) base(1) schema(1) major-version-2(2)}

-- Minor version: 1

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

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
  CrlSeries,
  EccP256CurvePoint,
  EciesP256EncryptedKey,
  EncryptionKey,
  GeographicRegion,
  GroupLinkageValue,
  HashAlgorithm,
  HashedId3,
  HashedId8,
```

```
  Hostname,
  IValue,
  LinkageValue,
  Opaque,
  Psid,
  PsidSsp,
  PsidSspRange,
  PublicEncryptionKey,
  PublicVerificationKey,
  SequenceOfHashedId3,
  SequenceOfPsidSsp,
  SequenceOfPsidSspRange,
  ServiceSpecificPermissions,
  Signature,
  SubjectAssurance,
  SymmetricEncryptionKey,
  ThreeDLocation,
  Time64,
  Uint3,
  Uint8,
  Uint16,
  Uint32,
  ValidityPeriod
FROM IEEE1609dot2BaseTypes {iso(1) identified-organization(3) ieee(111)
    standards-association-numbered-series-standards(2) wave-stds(1609)
    dot2(2) base(1) base-types(2) major-version-2 (2)}

;

--
--**********************************************************************
--
--  Structures for describing secured data
--
--**********************************************************************

-- Necessary to get certain tools to generate sample PDUs
-- TestIeee1609Dot2Data ::= Ieee1609Dot2Data
-- TestCertificate ::= Certificate

-- this structure belongs later in the file but putting it here avoids
-- compiler errors with certain tools
SignedDataPayload ::= SEQUENCE  {
    data              Ieee1609Dot2Data OPTIONAL,
    extDataHash       HashedData OPTIONAL,
    ...
}
    (WITH COMPONENTS {..., data PRESENT} |
     WITH COMPONENTS {..., extDataHash PRESENT})

Ieee1609Dot2Data ::= SEQUENCE  {
    protocolVersion    Uint8(3),
    content            Ieee1609Dot2Content
}

Ieee1609Dot2Content ::=  CHOICE  {
    unsecuredData            Opaque,
    signedData               SignedData,
    encryptedData            EncryptedData,
    signedCertificateRequest Opaque,
    ...
}

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

SignerIdentifier ::= CHOICE  {
    digest          HashedId8,
    certificate     SequenceOfCertificate,
    self            NULL,
    ...
}
```

```
ToBeSignedData ::= SEQUENCE  {
    payload         SignedDataPayload,
    headerInfo      HeaderInfo
}

HashedData::= CHOICE  {
    sha256HashedData  OCTET STRING (SIZE(32)),
    ...
}

HeaderInfo ::= SEQUENCE  {
    psid                    Psid,
    generationTime          Time64 OPTIONAL,
    expiryTime              Time64  OPTIONAL,
    generationLocation      ThreeDLocation OPTIONAL,
    p2pcdLearningRequest    HashedId3 OPTIONAL,
    missingCrlIdentifier    MissingCrlIdentifier OPTIONAL,
    encryptionKey           EncryptionKey OPTIONAL,
    ...,
    inlineP2pcdRequest      SequenceOfHashedId3 OPTIONAL,
    requestedCertificate    Certificate OPTIONAL
}

MissingCrlIdentifier ::= SEQUENCE  {
    cracaId         HashedId3,
    crlSeries       CrlSeries,
    ...
}

Countersignature ::= Ieee1609Dot2Data (WITH COMPONENTS {...,
  content (WITH COMPONENTS {...,
    signedData  (WITH COMPONENTS {...,
      tbsData (WITH COMPONENTS {...,
        payload (WITH COMPONENTS {...,
          data ABSENT,
          extDataHash PRESENT
        }),
        headerInfo(WITH COMPONENTS {...,
          generationTime PRESENT,
          expiryTime ABSENT,
          generationLocation ABSENT,
          p2pcdLearningRequest ABSENT,
          missingCrlIdentifier ABSENT,
          encryptionKey ABSENT
        })
      })
    })
  })
})

--**********************************************************************
--
--  Structures for describing encrypted data
--
--**********************************************************************


EncryptedData ::= SEQUENCE  {
    recipients     SequenceOfRecipientInfo,
    ciphertext     SymmetricCiphertext
}
RecipientInfo ::= CHOICE  {
    pskRecipInfo        PreSharedKeyRecipientInfo,
    symmRecipInfo       SymmRecipientInfo,
    certRecipInfo       PKRecipientInfo,
    signedDataRecipInfo PKRecipientInfo,
    rekRecipInfo        PKRecipientInfo
}

SequenceOfRecipientInfo ::= SEQUENCE OF RecipientInfo

PreSharedKeyRecipientInfo ::= HashedId8
SymmRecipientInfo ::= SEQUENCE  {
    recipientId        HashedId8,
    encKey             SymmetricCiphertext
}
```

```
PKRecipientInfo ::= SEQUENCE  {
    recipientId        HashedId8,
    encKey             EncryptedDataEncryptionKey
}

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

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

AesCcmCiphertext ::= SEQUENCE  {
    nonce           OCTET STRING (SIZE (12)),
    ccmCiphertext   Opaque -- 16 bytes longer than plaintext
}


--********************************************************************
--
--  Certificates and other security management data structures
--
--********************************************************************

-- Certificates are implicit (type = implicit, toBeSigned includes
-- reconstruction value, signature absent) or explicit (type = explicit,
-- toBeSigned includes verification key, signature present).

Certificate ::= CertificateBase (ImplicitCertificate | ExplicitCertificate)

SequenceOfCertificate ::= SEQUENCE OF Certificate

CertificateBase ::= SEQUENCE  {
    version                 Uint8(3),
    type                    CertificateType,
    issuer                  IssuerIdentifier,
    toBeSigned              ToBeSignedCertificate,
    signature               Signature OPTIONAL
}

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

ImplicitCertificate ::= CertificateBase (WITH COMPONENTS {...,
    type(implicit),
    toBeSigned(WITH COMPONENTS {...,
      verifyKeyIndicator(WITH COMPONENTS {reconstructionValue})
    }),
    signature ABSENT
    })

ExplicitCertificate ::= CertificateBase (WITH COMPONENTS {...,
    type(explicit),
    toBeSigned(WITH COMPONENTS {...,
        verifyKeyIndicator(WITH COMPONENTS {verificationKey})
    }),
    signature PRESENT
    })

IssuerIdentifier ::= CHOICE  {
    sha256AndDigest         HashedId8,
    self                    HashAlgorithm,
    ...,
    sha384AndDigest         HashedId8
}

ToBeSignedCertificate ::= SEQUENCE  {
    id                      CertificateId,
    cracaId                 HashedId3,
    crlSeries               CrlSeries,
    validityPeriod          ValidityPeriod,
```

```
    region                  GeographicRegion OPTIONAL,
    assuranceLevel          SubjectAssurance OPTIONAL,
    appPermissions          SequenceOfPsidSsp OPTIONAL,
    certIssuePermissions    SequenceOfPsidGroupPermissions OPTIONAL,
    certRequestPermissions SequenceOfPsidGroupPermissions OPTIONAL,
    canRequestRollover      NULL OPTIONAL,
    encryptionKey           PublicEncryptionKey OPTIONAL,
    verifyKeyIndicator      VerificationKeyIndicator,
    ...
}
(WITH COMPONENTS { ..., appPermissions PRESENT} |
 WITH COMPONENTS { ..., certIssuePermissions PRESENT} |
 WITH COMPONENTS { ..., certRequestPermissions PRESENT})

CertificateId ::= CHOICE  {
    linkageData             LinkageData,
    name                    Hostname,
    binaryId                OCTET STRING(SIZE(1..64)),
    none                    NULL,
    ...
}

LinkageData ::= SEQUENCE  {
    iCert                  IValue,
    linkage-value          LinkageValue,
    group-linkage-value    GroupLinkageValue OPTIONAL
}

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

PsidGroupPermissions ::= SEQUENCE  {
    subjectPermissions SubjectPermissions,
    minChainLength     INTEGER DEFAULT 1,
    chainLengthRange   INTEGER DEFAULT 0,
    eeType             EndEntityType DEFAULT {app}
}

SequenceOfPsidGroupPermissions ::= SEQUENCE OF PsidGroupPermissions

SubjectPermissions ::= CHOICE  {
    explicit        SequenceOfPsidSspRange,
    all             NULL,
    ...
}

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

END
```

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

-- Minor version: 1

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

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;


-- ---------------------------------------------------------------------
--
-- Integers
--
-- ---------------------------------------------------------------------
```

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

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

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


Opaque ::= OCTET STRING


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


-- ------------------------------------------------------------------
--
-- Time
--
-- ------------------------------------------------------------------

Time32 ::= Uint32
Time64 ::= Uint64

ValidityPeriod ::= SEQUENCE  {
    start          Time32,
    duration       Duration
}

Duration ::= CHOICE  {
    microseconds     Uint16,
    milliseconds     Uint16,
    seconds          Uint16,
    minutes          Uint16,
    hours            Uint16,
    sixtyHours       Uint16,
    years            Uint16
}

-- ------------------------------------------------------------------
--
-- Location
--
-- ------------------------------------------------------------------


GeographicRegion ::= CHOICE  {
    circularRegion      CircularRegion,
    rectangularRegion   SequenceOfRectangularRegion,
    polygonalRegion     PolygonalRegion,
    identifiedRegion    SequenceOfIdentifiedRegion,
    ...
}

CircularRegion ::= SEQUENCE  {
    center          TwoDLocation,
    radius          Uint16
}

RectangularRegion ::= SEQUENCE  {
    northWest       TwoDLocation,
    southEast       TwoDLocation
}

SequenceOfRectangularRegion ::= SEQUENCE OF RectangularRegion

PolygonalRegion ::= SEQUENCE SIZE(3..MAX) OF TwoDLocation
```

```
TwoDLocation ::= SEQUENCE  {
    latitude      Latitude,
    longitude     Longitude
}

IdentifiedRegion ::= CHOICE  {
    countryOnly           CountryOnly,
    countryAndRegions     CountryAndRegions,
    countryAndSubregions   CountryAndSubregions,
    ...
}

SequenceOfIdentifiedRegion ::= SEQUENCE OF IdentifiedRegion

CountryOnly ::= Uint16

CountryAndRegions ::= SEQUENCE  {
    countryOnly    CountryOnly,
    regions        SequenceOfUint8
}

CountryAndSubregions ::= SEQUENCE  {
    country             CountryOnly,
    regionAndSubregions  SequenceOfRegionAndSubregions
}

RegionAndSubregions ::= SEQUENCE  {
    region             Uint8,
    subregions         SequenceOfUint16
}

SequenceOfRegionAndSubregions ::= SEQUENCE OF RegionAndSubregions

ThreeDLocation ::= SEQUENCE  {
    latitude        Latitude,
    longitude       Longitude,
    elevation       Elevation
}

Latitude ::= NinetyDegreeInt
Longitude ::= OneEightyDegreeInt
Elevation ::= ElevInt

NinetyDegreeInt ::= INTEGER  {
    min         (-900000000),
    max         (900000000),
    unknown     (900000001)
} (-900000000..900000001)

KnownLatitude ::= NinetyDegreeInt (min..max) -- Minus 90deg to +90deg in microdegree intervals
UnknownLatitude ::= NinetyDegreeInt (unknown)

OneEightyDegreeInt ::= INTEGER  {
    min         (-1799999999),
    max         (1800000000),
    unknown     (1800000001)
} (-1799999999..1800000001)

KnownLongitude ::= OneEightyDegreeInt (min..max)
UnknownLongitude ::= OneEightyDegreeInt (unknown)

ElevInt ::= Uint16 -- Range is from -4096 to 61439 in units of one-tenth of a meter

-- ----------------------------------------------------------------
--
-- Crypto
--
-- ----------------------------------------------------------------

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

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

```
    sSig     OCTET STRING (SIZE (32))
}

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

EccP256CurvePoint ::= CHOICE  {
    x-only           OCTET STRING (SIZE (32)),
    fill             NULL,  -- consistency with 1363 / X9.62
    compressed-y-0   OCTET STRING (SIZE (32)),
    compressed-y-1   OCTET STRING (SIZE (32)),
    uncompressedP256 SEQUENCE  {
        x OCTET STRING (SIZE (32)),
        y OCTET STRING (SIZE (32))
    }
}

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


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

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

EciesP256EncryptedKey ::= SEQUENCE  {
    v               EccP256CurvePoint,
    c               OCTET STRING (SIZE (16)),
    t               OCTET STRING (SIZE (16))
}

EncryptionKey ::= CHOICE  {
    public               PublicEncryptionKey,
    symmetric            SymmetricEncryptionKey
}

PublicEncryptionKey ::= SEQUENCE  {
    supportedSymmAlg      SymmAlgorithm,
    publicKey             BasePublicEncryptionKey
}

BasePublicEncryptionKey ::= CHOICE  {
    eciesNistP256             EccP256CurvePoint,
    eciesBrainpoolP256r1      EccP256CurvePoint,
    ...
}

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

SymmetricEncryptionKey ::= CHOICE  {
    aes128Ccm     OCTET STRING(SIZE(16)),
    ...
}

-- ---------------------------------------------------------------
--
```

```
-- PSID / ITS-AID
--
-- ------------------------------------------------------------------


PsidSsp ::= SEQUENCE  {
    psid                    Psid,
    ssp                     ServiceSpecificPermissions OPTIONAL
}

SequenceOfPsidSsp ::= SEQUENCE OF PsidSsp

Psid ::= INTEGER (0..MAX)

SequenceOfPsid ::= SEQUENCE OF Psid

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

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

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

SequenceOfPsidSspRange ::= SEQUENCE OF PsidSspRange

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

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

SequenceOfOctetString ::= SEQUENCE (SIZE (0..MAX)) OF
    OCTET STRING (SIZE(0..MAX))


-- ------------------------------------------------------------------
--
-- Goes in certs
--
-- ------------------------------------------------------------------

SubjectAssurance ::= OCTET STRING (SIZE(1))

CrlSeries ::= Uint16


-- ------------------------------------------------------------------
--
-- Pseudonym Linkage
--
-- ------------------------------------------------------------------

IValue ::= Uint16
Hostname ::= UTF8String (SIZE(0..255))
LinkageValue ::= OCTET STRING (SIZE(9))
GroupLinkageValue ::= SEQUENCE  {
    jValue  OCTET STRING (SIZE(4)),
    value   OCTET STRING (SIZE(9))
}

LaId ::= OCTET STRING (SIZE(2))
LinkageSeed ::= OCTET STRING (SIZE(16))

END
```

History

Document history		
V1.1.1	April 2013	Publication
V1.2.1	June 2015	Publication
V1.3.1	October 2017	Publication