PKI architecture and technical specifications (v1.1)

This is the second deliverable of task 2 (ISE Project).

July 2015

IRT SystemX
8, avenue de la Vauve
91120 Palaiseau
contact@irt-systemx.fr
www.irt-systemx.fr
Tél. : +33 (0)1 69 08 05 68

PUBLIC

Revision History

Version number	Update date	Main authors	Summary of changes
1.0	26/06/15	Erwann ABALEA (IDNOMIC) Hafeda BAKHTI (IDNOMIC) Rémi BLANCHER (IDNOMIC) Benoît CHARLES (IDNOMIC) Brigitte LONC (RENAULT)	First stable version
1.1	15/07/15	Erwann ABALEA (IDNOMIC) Hafeda BAKHTI (IDNOMIC)	Additions in ASN.1 module

Table of contents

Revision History 1

1. Introduction 4

1.1 Intended audience 4

1.2 Typographic conventions 4

1.3 Definitions and abbreviations 4

1.4 References 6

1.4.1 Normative references 6

1.4.2 Informative references 6

1.5 Objectives 7

1.5.1 Security objectives 7

2. System overview 8

2.1 High level architecture 8

2.2 Description of roles 8

2.2.1 Operator 8

2.2.2 Manufacturer 8

2.2.3 ITS-Station 8

2.3 Protocol definitions 9

3. PKI System 10

3.1 Functions 10

3.1.1 Root Certificate Authority (RCA) component functions 10

3.1.2 Enrolment Authority (EA) component functions 12

3.1.3 Authorization Authority (AA) component functions 15

3.1.4 Distribution Center (DC) component functions 16

3.2 Data structures 17

3.2.1 General design rules 17

3.2.2 Data type 17

3.2.3 Algorithm identifier types 17

3.2.4 SignedData type 18

3.2.5 EncryptedData type 20

3.2.6 Certificate Revocation List 21
3.2.7 Trust-service Status List 21
3.2.8 Mapping with ETSI Standards 23
3.3 Requests 24
3.3.1 Create RCA certificate 24
3.3.2 Create Authority (EA/AA) certificate 24
3.3.3 Request Enrolment Credential (EC) 24
3.3.4 Request Authorization Ticket (AT) 27
3.3.5 Validate Authorization Ticket (AT) request 31
3.3.6 Get CRL 35
3.3.7 Get TSL 35
Appendix A: Examples of request 36
Appendix B: Encryption of a message 42
Appendix C: ASN.1 module 43

1. Introduction

1.1 Intended audience

This document is primarily written for the implementers. The document provides references to the high level PKI architecture and directs the reader to the detailed information cited in the first deliverable of task 2: PKI System Requirements Specifications.

1.2 Typographic conventions

The following typographic conventions are used in this document:

`EX ::= SEQUENCE {}`	Code example
[1]	Numbers in-between square brackets are references to publications mentioned in the appendix References.

1.3 Definitions and abbreviations

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

	Synonyms	Definitions
Access Point		Access point is a HTTP URL used to access web services of the PKI.
Anonymity		Anonymity is the ability of a user to use a resource or service without disclosing its identity.
Authorization Authority (AA)	Pseudonym Certificate Authority (PCA)	Security management entity responsible for issuing, monitoring the use of authorization tickets.
Authorization Ticket (AT)	Pseudonym Certificate (PC)	Data object that demonstrates that the holder has permissions which entitle him to take specific actions.
Confidentiality		Confidentiality is a set of rules or a promise that limits access or places restrictions on certain types of information.
Certificate Revocation List (CRL)		Certificate Revocation List is a list digitally signed by a CA that contains certificates identities that are no longer valid.

Distribution Center (DC)		Distribution Center provides ITS-S the updated trust information necessary for performing the validation process to control that received information is coming from a legitimate and authorized ITS-S or PKI certification authority.
Enrolment Authority (EA)	Long Term Certificate Authority (LTCA)	Security management entity responsible for the life cycle management of enrolment credentials.
Enrolment Credential (EC)	Long Term Certificate (LTC)	Data object that is used in message exchanges between an ITS Station and a security management entity and demonstrates that the valid holder is entitled to apply for authorization tickets.
EA identifier		EA identifier is composed of the last 8 octets of the SHA256 digest of the EA certificate. This identifier is stored as a `HashedId8` data type (see [1]).
EC identifier		EC identifier is composed of the last 8 octets of the SHA256 digest of the EC certificate. This identifier is stored as a `HashedId8` data type (see [1]).
Integrity		Integrity means maintaining and assuring the accuracy and consistency of data over its entire life-cycle.
ITS Station (ITS-S)		ITS Station is end-user of the PKI system. The PKI system provides it different certificates (EC or AT) to allow secure communications. ITS-S can be normal vehicles, public safety vehicles, roadside stations, nomadic devices and traffic management centers…
Manufacturer		Manufacturer installs necessary information for security management in ITS-S at production.
Root CA (RCA)		Root Certificate Authority is the root of trust for all certificates within the PKI hierarchy. Root CA issues certificates for EAs and AAs

		to authorize them to issue certificates to end-entities. It also defines and controls policies among all certificate issuers. The Root CA is required when a new EA or AA shall be created, or when the lifetime of EA or AA certificate expires.
Trust-service Status List (TSL)		The Trust-service Status List is a signed list which contains new RCA certificates, EA and AA certificates and PKI service addresses (AA and DC). This list is signed by the RCA and can be transmitted over the air.

1.4 References

1.4.1 Normative references

The following references documents are not essential to the use of the present document but they assist the user with regard to a particular subject area.

[1] ETSI TS 103 097 (v1.2.1): ITS; Security; Security header and certificate formats

[2] ETSI TS 102 941 (v1.1.1): ITS; Security; Trust and Privacy Management

[3] X.680: Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation

[4] X.690: Information Technology - ASN.1 encoding rules: Specifications of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)

[5] X.691: Information Technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)

[6] RFC2616: HTTP/1.1

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

[8] ETSI EN 302637-2: ITS; Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service

[9] ETSI EN 302637-3: ITS; Vehicular Communications; Basic Set of Applications; Part 2: Specifications of Decentralized Environmental Notification Basic Service

[10] ETSI TR 102 965: ITS; Application Object Identifier (ITS-AID); Registration list

[11] FIPS 198-1: The Keyed-Hash Message Authentication Code (HMAC)

1.4.2 Informative references

The following references documents are not essential to the use of the present document but they assist the user with regard to a particular subject area.

[i.1] PKI System Requirements Specifications (ISX-TEO-SE-ISE-LIV-0042) [i.2] RFC5246: The TLS Protocol version 1.2

[i.3] RFC5084: Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)

[i.4] SEC 1: Elliptic Curve Cryptography version 2.0

[i.5] ETSI TS 102 860: Intelligent Transport Systems (ITS); Classification and management of ITS application objects

1.5 Objectives

This document describes the functionalities of the PKI system for ISE project.

The PKI system is divided into four entities:

- The Root Certificate Authority for the generation of CA private keys, a key step in the initiation of a trust chain.
- The Enrollment Authority (EA), used by Manufacturer and ITS-S, respectively for the ITS-S lifecycle management and for the provisioning of ECs.
- The Authorization Authority (AA) used by ITS-S, for requesting ATs.
- The Distribution Center, used by ITS-S to retrieve CRL and TSL.

The PKI for ITS-S is a set of software modules enabling distribution of certificates for secured communication between ITS-S.

1.5.1 Security objectives

The hereafter described protocol tries to reach the following security objectives:

- **Authentication/authorization control**: authentication consists to be sure of the identity which sends data. Authorization control is the verification of an access policy, based on a trusted authentication. Authenticate all entities participating in the protocol is required to prevent illegitimate persons to enter in the system, or to access some unauthorized resources or services.

- **Integrity**: the integrity of all transmitted data is important to ensure that the contents of the received data are not altered.

- **Confidentiality/Privacy**: data should only be accessed by authorized entities. The real identity of ITS Station has to be protected, by cryptographic mechanisms and depending on the type of data sent.

- **Non-repudiation/Traceability**: Non-repudiation is necessary to prevent ITS Station or others entities from denying the transmission or the content of their messages. Traceability, which is the warranty that an entity can’t refute the emission or reception of information, is also extremely important.

- **Unlinkability:** ability of a user to make multiple uses of resources or services without others being able to link these uses together.

- **Anonymity:** ability of a user to use a resource or service without disclosing the user's identity.

2. System overview

2.1 High level architecture

Figure 1 shows the PKI high level architecture that was the starting point in task 2 of ISE project.

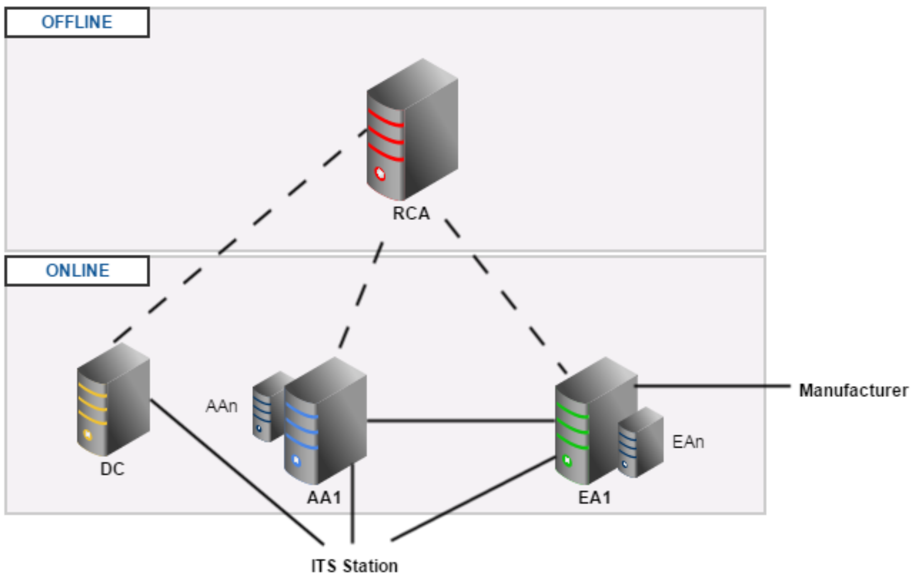

Figure 1

2.2 Description of roles

2.2.1 Operator

The “operator” role is to install and update necessary information for security management in ITS-S during operation.

2.2.2 Manufacturer

The “manufacturer” role is to install necessary information for security management in ITS-S at production. At least, the manufacturer bootstraps the process for manufacturing a trusted ITS-S in production site:

- generates and stores securely required crypto-material in its security module,
- installs the RCA certificate,
- initializes EA certificate and network address,
- configures a DC network address.

2.2.3 ITS-Station

The “ITS-Station” role is to request certificates (ECs and ATs) from EA and AA. ITS Station only has access to the web service interface.

2.3 Protocol definitions

To support security management of trusted ITS-S (vehicles, road-side or center stations), an automatic communication means with the different PKI modules shall be provided by the ITS-S embedded system. This section specifies the higher layers of the protocol stack and assumes either a fixed or cellular network with the ITS-S or an ITS G5 communication profile supporting IP connectivity (e.g. GN6, IPv6 /LLC and SNAP as specified in IEEE 1609.3).

Machine-to-machine communications with the EA, AA, and DC components use HTTP/1.1 as a transport mechanism, over TCP, over IP. No supplementary cryptographic layer such as TLS is required.

Messages are sent as HTTP GET or POST requests. Parameters for the POST requests and responses, and complete path for GET requests are described in the corresponding messages descriptions.

The notation language used hereafter is ASN.1 defined in [3].

The preferred encoding rules are ASN.1 DER (Distinguished Encoding Rules), defined in [4].

ASN.1 DER is usually used in trusted services, for example X.509 certificate, e-passport, eIDAS, time stamping, S/MIME, etc.

Key points in favor of DER:

- Canonicalization is mandatory for signature
- DER encoding/decoding routines are available in most programming languages
- DER encoded structures are easy to analyze and debug

Human-to-machine communications with the EA and AA use HTTP/1.1 as a transport mechanism, over TCP, over IP, with TLS. A web interface (used by operators and manufacturers) is intended: this is out of scope of this document.

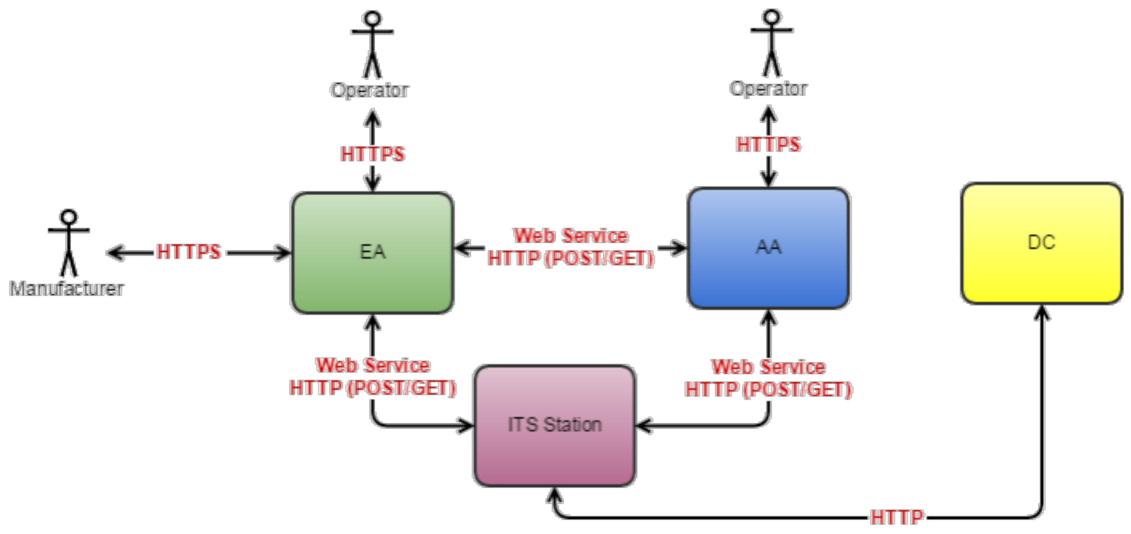

Figure 2

3. PKI System

3.1 Functions

3.1.1 Root Certificate Authority (RCA) component functions

The features of RCA component (described in the first deliverable “PKI System Requirements Specifications”) are:

- Creation of RCA key pair and self-signed certificate;
- Issuance of CA (EA or AA) certificates;
- Revocation of CA (EA or AA) certificates;
- Generation of CA CRL;
- Generation of TSL.

3.1.1.1 Create a RCA certificate

<u>Objective</u>

Create a RCA certificate.

<u>Input Data</u>

The following information is provided:

- The assurance level
- The ITS AID list
- The validity restrictions
    - The date (`time_start_and_end`)
    - The region (optional)

<u>The name of the Certificate Authority (optional)Output Data</u>

A RCA certificate is created. The format of this certificate is described in ETSI Standard, see [1].

<u>Traceability</u>

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

3.1.1.2 Create an EA certificate

<u>Objective</u>

Create an EA certificate.

<u>Input Data</u>

The following information is provided:

- The public keys (verification and encryption) to be signed
- The ITS AID list in accordance with ITS AID list of RCA

    - The assurance level
    - The validity restrictions
        - The date (`time_start_and_end`)
        - The region in accordance with RCA’s region (if applicable)

<u>The name of the Certificate Authority (optional)Output Data</u>

An EA certificate is created. The format of this certificate is described in ETSI Standard, see [1].

<u>Traceability</u>

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

3.1.1.3 Create an AA certificate

<u>Objective</u>

Create an AA certificate.

<u>Input Data</u>

The following information is provided:

    - The public key (verification key and encryption key) to signed
    - The assurance level
    - The ITS AID list in accordance with ITS AID list of RCA
    - The validity restrictions
        - The date (`time_start_and_end`)
        - The region in accordance with RCA’s region (if applicable)

<u>The name of the Certificate Authority (optional)Output Data</u>

An AA certificate is created. The format of this certificate is described in ETSI Standard, see [1].

<u>Traceability</u>

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

3.1.1.4 Revoke a CA certificate

<u>Objective</u>

Revoke a CA certificate (EA or AA).

<u>Input Data</u>

The following information is provided:

  - EA or AA certificate to be revoked

<u>Output Data</u>

A success response is sent.

<u>Traceability</u>

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

3.1.1.5 Generate CA Certificate Revocation List (CRL)

Objective

Generate CA Certificate Revocation List.

Input Data

The following information is provided:

- List of revoked certificates

Output Data

The CA CRL is generated. The format of the CA CRL is described in 3.2.6.

Traceability

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

3.1.1.6 Generate Trust-service Status List (TSL)

Objective

Generate the Trust-service Status List.

Input Data

The following information is provided:

- CAs certificates
- PKI services addresses (RCA, EA, AA and DC)

Output Data

The TSL is generated. The format of the TSL is described in 3.2.7.

Traceability

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

3.1.2 Enrolment Authority (EA) component functions

The features of EA component (described in the first deliverable “PKI System Requirements Specifications”) are:

- Registration of ITS-S
- Management of ITS-S status
- Management of ITS-S permissions
- Issuance of Enrolment Credentials
- Verification of ITS-S permissions for AT request

3.1.2.1 Register ITS Station

This feature is executed directly by the manufacturer through a graphical user interface (GUI).

3.1.2.2 Change status of ITS Station

This feature is executed directly by the manufacturer or the operator through a graphical user interface.

3.1.2.3 Change permissions of ITS Station

This feature is executed directly by the manufacturer through a graphical user interface.

3.1.2.4 Request Enrolment Credential

Role(s)

Only the ITS Station possessing the appropriate elements can perform this action.

Objective

An ITS Station requests an enrolment credential (EC).

Input Data

ITS Station provides the following information:

- The canonical identifier of ITS Station
- The public key (verification key)
- The response decryption public key
- The ITS AID SSP List (see [8] and [9])
- The validity restrictions (optional)
    - The date(s)
    - The region

Output Data

EA returns a message containing:

- An EC, the format of this certificate is described in ETSI Standard, see [1].
- A response code (see 3.3.3.2 for more information).

Possible errors

For each of the errors below, an error message is returned to ITS Station responsible for the action.

- ITS Station fails to provide the required values in the request
- ITS Station is unknown (not registered)
- An internal error occurs
- Etc.

Traceability

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

The traceability of this action is mandatory to lift the anonymity of ITS station.

3.1.2.5 Validate Authorization Ticket (AT) request

Role(s)

Any AA can perform this operation.

Objective

Validate AT request before producing an AT to the relevant ITS-S.

Input Data

The AA provides the following information as below to the EA for authenticating the requesting ITS-S and checking its permissions to get requested Authorization Ticket:

- EA identifier
- Validity restrictions
  - The date(s)
  - The region (optional)
- Subject attributes
- Encrypted structure containing the signature and the EC identifier

Output Data

EA returns a message containing:

- A response code (see 3.3.5.2 for more information).

Possible errors

- The ITS-S is not authorized to get pseudonym certificates
- The ITS-S is not managed by the EA
- Etc.

Traceability

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

3.1.3 Authorization Authority (AA) component functions

The features of AA component (described in the first deliverable “PKI System Requirements Specifications”) are:

- Issuance of Authorization Tickets.

3.1.3.1 Request authorization ticket(s) (AT)

Role(s)

Any ITS possessing an EC can request AT.

Objective

ITS station requests AT.

Input Data

ITS Station provides the following information:

- Verification public key(s)
- Encryption public key(s)
- EA identifier
- Validity restrictions

    - The date(s)
    - The region (optional)
  - Subject attributes

Output Data

AA returns a message containing:

- An AT, the format of this certificate is described in ETSI Standard, see [1].
- A response code (see 3.3.4.23.3.5.2 for more information).

Possible errors

For each of the errors below, an error message is returned to ITS -S responsible for the action.

- ITS-S fails to provide the required values in the request.
- EA can't be reached.
- EA is unable to verify permissions of relevant ITS Station (see Validate AT request function).
- An internal error occurs.
- Etc.

Traceability

The action is entered in the audit log.

The action is viewable in the log from the operator interface.

The traceability of this action is mandatory to lift the anonymity of ITS station.

3.1.4 Distribution Center (DC) component functions

The features of DC (described in the first deliverable "PKI System Requirements Specifications") are:

- Publication of TSL
- Publication of CA CRL

3.1.4.1 Get CA Certificate Revocation List

Role(s)

Everybody can perform this operation.

Objective

Everybody retrieves an updated CRL.

Output Data

The DC provides the CRL. The format of this CRL is described in 3.2.6.

Possible errors

For each of the errors below, an error message is returned to ITS Station responsible for the action.

- An internal error occurs.
- Etc.

3.1.4.2 Get Trust-service Status List

Role(s)

Everybody can perform this operation.

Objective

Everybody retrieves an updated Trust-service Status List.

Output Data

The DC provides the TSL. The format of this TSL is described in 0.

Possible errors

For each of the errors below, an error message is returned to ITS-S responsible for the action.

- An internal error occurs
- Etc.

3.2 Data structures

The data structures `Data`, `SignedData`, `EncryptedData` and associated algorithm identifiers types described below are used to build protocol messages between ITS-S and PKI, and between PKI entities, with clearly defined security properties.

The CRL structure allows the revocation of long duration certificates used by actors and PKI entities.

3.2.1 General design rules

- `version` is placed first to allow for the block format to change (should not be used to describe the version of the inner content)
- `contentType` describes what is to be found in the associated inner content (and its version)
- cryptographic parameters are before the data to decrypt/verify (hash/signature algorithm, recipients, encryptionParameters), this allows to stream data
- `signature` is placed after the data

3.2.2 Data type

-- used as the most external container

The content is optional to allow for external content declaration

```
        Data ::= SEQUENCE {
            version Version DEFAULT v1,
            contentType ContentType,
            content OCTET STRING OPTIONAL }
```

```
        ContentType ::= OBJECT IDENTIFIER
```

3.2.3 Algorithm identifier types

This section defines sets of algorithms:

- signature algorithms
- data encryption algorithms
- key encryption algorithms
- hash algorithms

Each defined algorithm is associated to a unique identifier and is accompanied by optional parameters where applicable. The sets of algorithms are dynamically extensible (at runtime), which allows for crypto agility.

```
        SignatureAlgorithmIdentifier ::= SEQUENCE {
            algorithm ALGORITHM.&id({SignatureFunctions}),
            parameters ALGORITHM.&Type({SignatureFunctions}{@algorithm}) OPTIONAL }
```

```
        ContentEncryptionAlgorithmIdentifier ::= SEQUENCE {
            algorithm ALGORITHM.&id({DataEncryptionFunctions}),
            parameters ALGORITHM.&Type({DataEncryptionFunctions}{@algorithm}) OPTIONAL }
```

```
        HashAlgorithmIdentifier ::= SEQUENCE {
            algorithm ALGORITHM.&id({HashFunctions}),
            parameters ALGORITHM.&Type({HashFunctions}{@algorithm}) OPTIONAL }
```

```
        KeyEncryptionAlgorithmIdentifier ::= SEQUENCE {
            algorithm ALGORITHM.&id({KeyEncryptionFunctions}),
            parameters ALGORITHM.&Type({KeyEncryptionFunctions}{@algorithm}) OPTIONAL }
```

3.2.4 SignedData type

This data structure is flexible enough to allow for internal or external signed content, multiple signers, multiple signatures, and one-pass verification (stream).

Data is signed using the following process:

- an empty `SignedData` structure is created, with `version` set to v1, and `signedContentType` set to the appropriate value
- the signed data can either be enclosed in an OCTET STRING and included in the `SignedData` structure, or left aside (detached or external signature)
- each signer does:
    - choose the preferred hash algorithms: one to digest the signed content, one to digest the attributes
    - optionally include those hash algorithm identifiers in the `hashAlgorithms` collection, in order to facilitate the one-pass signature verification
    - digest the signed content and store the result in an `Attribute` structure of type `attr-messageDigest`
    - create an `Attribute` structure of type `attr-contentType` containing

      the `signedContentType` value
    - create a `SignerInfo` structure containing:
        - the 2 precedent `Attribute` structures in the `signedAttributes` collection

        - an optional `Attribute` of type `attr-signingTime` in the `signedAttributes` collection
        - the `signerIdentifier` set to the appropriate value
        - optionally the certificate chain in order to validate the signer
        - the `digestAlgorithm` equal to the hash algorithm used to digest the signed content
        - the `signatureAlgorithm` set to the signature algorithm used by the signer
        - the `signature` value, result of the signature operation applied to the serialization of the `signedAttributes` structure
    - include the composed `SignerInfo` structure in the `signerInfos` collection

Three different attributes are defined:

- `attr-contentType`: its type is a `ContentType`, its value shall be identical to the `signedContentType` of the SignedData structure
- `attr-messageDigest`: its type is an OCTET STRING, its value shall be the result of processing the `signedContent` octets through the hash algorithm identified by `digestAlgorithm`
- `attr-signingTime`: its type is a Time32, its value is the number of seconds elapsed since 01 January 2004 00:00:00 UTC, on the TAI time scale; i.e. its semantic is identical to the Time32 data type defined by ETSI Standard [1] and IEEE 1609.2v2

It is important that the `attr-messageDigest` and `attr-contentType` attributes are included in the `signedAttributes`. Their presence is mandatory. The `attr-signingTime` is optional, and can be required depending on the context.

```
       SignedData ::= SEQUENCE {
           version Version DEFAULT v1,
           hashAlgorithms HashAlgorithmsIdentifiers,
           signedContentType ContentType,
           signedContent OCTET STRING OPTIONAL,
           signerInfos SignerInfos }
```

```
       HashAlgorithmsIdentifiers ::= SEQUENCE OF HashAlgorithmIdentifier
```

```
       SignerInfos ::= SEQUENCE OF SignerInfo
```

```
       SignerInfo ::= SEQUENCE {
           version Version DEFAULT v1,
           signer [0] SignerIdentifier DEFAULT self:NULL,
           digestAlgorithm [1] HashAlgorithmIdentifier DEFAULT { algorithm id-sha256 },
           signatureAlgorithm [2] SignatureAlgorithmIdentifier DEFAULT { algorithm ecdsa-
with-SHA256 },
           signedAttributes SignedAttributes,
           certificateChain SEQUENCE OF Certificate OPTIONAL,
           signature SignatureValue }
```

```
       SignerIdentifier ::= CHOICE {
           self NULL,
           certificateDigest CertificateDigest,
           certificate Certificate }
```

```
       CertificateDigest ::= SEQUENCE {
        algorithm HashAlgorithmIdentifier DEFAULT { algorithm id-sha256 },
        digest HashedId8 }
```

```
    SignedAttributes ::= SEQUENCE OF Attribute
```

```
    Attribute ::= SEQUENCE {
        attrType ATTRIBUTE.&id({SupportedAttributes}),
        attrValue ATTRIBUTE.&Type({SupportedAttributes}{@attrType}) OPTIONAL }
```

```
    SignatureValue ::= OCTET STRING
```

-- `SignatureValue` should be opaque to the user/caller of security functions.
-- Internally, an ECDSA signature contains the following structure:

```
    Ecdsa-Sig-Value ::= SEQUENCE {
        r INTEGER,
        s INTEGER }
```

3.2.5 EncryptedData type

Data is encrypted to a number of recipients following this process:

- The sender chooses a content encryption algorithm and parameters.
- The sender randomly generates a content encryption symmetric key
- The sender encrypts this content encryption symmetric key for each recipient
- For each recipient, a corresponding `RecipientInfo` structure is built
- The content is encrypted using chosen algorithm, parameters, and content encryption symmetric key
- The encrypted content, encryption algorithm parameters, and all `RecipientInfo` instances are collected together to form an `EncryptedData` structure

When the recipient is identified by its public key and not by its certificate(for example when the recipient requests a certificate), the `recipients` field of type `HashedId8` shall be calculated as the 8 lowest order octets of the SHA256 digest of the encoded public key in compressed form.

If the encrypted content is to be transmitted outside of this `EncryptedData` structure, the `EncryptedData` structure can be used to transport the encrypted symmetric encryption key and encryption parameters. The `encryptedContent` element is optional.

```
    EncryptedData ::= SEQUENCE {
        version Version DEFAULT v1,
        recipients RecipientInfos,
        encryptedContentType ContentType,
        encryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
        encryptedContent OCTET STRING OPTIONAL }
```

```
    RecipientInfos ::= SEQUENCE SIZE (1..MAX) OF RecipientInfo
```

```
        RecipientInfo ::= SEQUENCE {
            recipient HashedId8,
            kexalgid KeyEncryptionAlgorithmIdentifier DEFAULT { algorithm id-ecies-103097 },
            encryptedKeyMaterial OCTET STRING }
```

If `kexalgid` is the algorithm identified by `id-ecies-103097`, then the `encryptedKeyMaterial` shall contain the serialization of an `ECIESEncryptedKey103097` data type.

3.2.6 Certificate Revocation List

The Certificate Revocation List (CRL) is generated and signed by the RCA component.

ASN.1 notation definition

```
        Crl ::= SEQUENCE {
            unsignedCrl ToBeSignedCrl,
            signatureAlgorithm SignatureAlgorithmIdentifier,
            signature Signature } -- signature is applied on unsignedCrl
```

```
        ToBeSignedCrl ::= SEQUENCE {
            version Version,
            signer SignerIdentifier,
            thisUpdate Time32,
            nextUpdate Time32,
            entries SEQUENCE OF HashedId8 }
```

The choice between HashedId8 and HashedId10/CertId10 should be dictated by the collision probability of 8/10 bytes reduced digests of certificates.

Entry Length	Number of certificates	Collision Probability
8	2^24	0.00076%
8	2^30	3.07668%
8	2^31	11.75031%
8	2^32	39.34693%
10	2^32	0.00076%
10	2^38	3.07668%
10	2^39	11.75031%
10	2^40	39.34693%

Figure 3

3.2.7 Trust-service Status List

ASN.1 notation definition

```
        Tsl ::= SEQUENCE {
            unsignedTsl ToBeSignedTsl,
            signatureAlgorithm SignatureAlgorithmIdentifier,
            signature SignatureValue }
        -- signature is applied on unsignedTsl
```

```
        ToBeSignedTsl ::= SEQUENCE {
```

```
        version Version,
        signerInfo SignerIdentifier,
        notBefore Time32,
        notAfter Time32,
        trustServices SEQUENCE OF TrustService }
```

```
    TrustService ::= SEQUENCE {
        serviceId TRUSTSERVICE.&id ({TrustServiceSet}),
        serviceValue TRUSTSERVICE.&Value ({TrustServiceSet}{@serviceId}) }
```

```
    TrustServiceSet TRUSTSERVICE ::=
        {   ts-foreignRoot
          | ts-renewedRoot
          | ts-ea
          | ts-aa
          | ts-distributionCenter
          | ts-otherTslPointer
          , ... }
```

```
    TRUSTSERVICE ::= CLASS {
        &id ServiceType UNIQUE,
        &Value }
    WITH SYNTAX {
        SYNTAX &Value
        ID &id }
```

```
    ts-foreignRoot TRUSTSERVICE ::= {
        SYNTAX Certificate
        ID foreignRoot }
```

```
    ts-renewedRoot TRUSTSERVICE ::= {
        SYNTAX SEQUENCE {
            rootCertificate Certificate,
            linkRootCertificate Certificate }
        ID renewedRoot }
```

```
    ts-ea TRUSTSERVICE ::= {
        SYNTAX SEQUENCE {
            certificate Certificate,
            linkedCertificate Certificate OPTIONAL,
            accessPoint IA5String }
        ID ea }
```

```
    ts-aa TRUSTSERVICE ::= {
        SYNTAX SEQUENCE {
            certificate Certificate,
            accessPoint IA5String}
        ID aa }
```

```
    ts-distributionCenter TRUSTSERVICE ::= {
        SYNTAX IA5String
        ID distributionCenter }
    ts-otherTslPointer TRUSTSERVICE ::= {
        SYNTAX IA5String
        ID otherTslPointer }
```

```
        ServiceType ::= ENUMERATED {
            foreignRoot,
            renewedRoot,
            ea,
            aa,
            distributionCenter,
            otherTslPointer,
            ... }
```

3.2.8 Mapping with ETSI Standards

Some data types defined in ETSI Standard [1] and used in this protocol need to be redefined in ASN.1 notation:

```
        HashedId8 ::= OCTET STRING (SIZE(8))
        Certificate ::= OCTET STRING
        Time32 ::= INTEGER (0..4294967295)
```

The types `SubjectAttribute`, `ValidityRestriction`, `verification_key` and `its_aid_ssp_list` are defined in ETSI Standard [1].

A vector of `SubjectAttribute` elements as used by this protocol will be represented by the `SubjectAttributes` type. The content of an element of this data type will be the binary serialization of a variable-length vector with variable-length length encoding of `SubjectAttribute` elements. Similarly, a vector of `ValidityRestriction` elements will be represented by the `ValidityRestrictions` type, and the content of an element of this data type will be the binary serialization of a variable-length vector with variable-length length encoding of `ValidityRestriction` elements.

```
        SubjectAttributes ::= OCTET STRING
        ValidityRestrictions ::= OCTET STRING
```

For example, a vector of 2 `SubjectAttribute` elements (a `verification_key` and an `its_aid_ssp_list` composed of 2 ITS-AID-SSP) will be encoded as the octet string 30000002C43CDA0AD74CC8A93141DBE4F2C353EDB8DD416DB14F1766A638E00B7EE2A752210B24030 10000250401000000", which is decomposed as:

```
        30 (variable-length length of the vector)
        {
          00 (type=verification_key)
          { <PublicKey>
            00 (algorithm=ecdsa_nistp256_with_sha256)
            { <EccPoint>
              02 (type=compressed_lsb_y_0)
              C43CDA0AD74CC8A93141DBE4F2C353EDB8DD416DB14F1766A638E00B7EE2A752 (x)
            }
          }
          21 (type=its_aid_ssp_list)
          {
            0B (variable-length length of the vector)
            {
              24 (its_aid=CAM)
              03 (variable-length length of the SSP)
              010000 (service_specific_permissions)
              25 (its_aid=DENM)
              04 (variable-length length of the SSP)
              01000000 (service_specific_permissions)
            }
        }
      }
```

3.3 Requests

3.3.1 Create RCA certificate

RCA generates its key pair and generates its self-signed certificate under trusted roles control.

3.3.2 Create Authority (EA/AA) certificate

EA and AA requests are transmitted by an off-band mechanism to the RCA entity.

3.3.2.1 Request format

`ITSCertificateRequest` data type defines a standalone certificate request, which can be used to transport EA or AA certificate request to the RCA.

```
      ITSCertificateRequest ::= SEQUENCE {
      itsCertReq ITSCertificateRequestContent,
      signatureAlgorithm SignatureAlgorithmIdentifier DEFAULT { algorithm ecdsa-with-SHA256
},
      signature SignatureValue }
```

```
      ITSCertificateRequestContent ::= SEQUENCE {
      version Version DEFAULT v1,
      subjectName OCTET STRING (SIZE(0..32)),
      subjectAttributes OCTET STRING,
      validityRestrictions OCTET STRING }
```

The following profile shall apply:

- `version` is set to v1 (0)
- `subjectAttributes` shall contain the serialization of a subject_attributes data type and shall contain both a `verification_key` and an `encryption_key` elements
- `validityRestrictions` shall contain the serialization of the `validity_restrictions` data type
- the signature is applied to the `itsCertReq` field using the private key corresponding to the public key declared as `verification_key` (i.e. the request is self-signed)

`subject_attributes` and `validity_restrictions` are defined in [1].

3.3.3 Request Enrolment Credential (EC)

POST http://<ea_access_point>

Inputs:

• Content-type: application/x-its-request

• Content: binary encoded `EnrolmentRequest` object

Outputs:

• Content-type: application/x-its-response

• Content: binary encoded `EnrolmentResponse` object

3.3.3.1 Request format

The ITS-S must build its EC request by following this process:

- an ECC private key is randomly generated (the response-decryption-key), the corresponding public key is computed (response-encryption-key)
- an `InnerECRequest` structure is built, containing:
    - a randomly generated `requestIdentifier`
    - the canonical identifier of the ITS-S
    - the desired attributes
    - some optional restrictions
    - the response-encryption-key
- a `SignedData` structure is built, with:
    - the `signedContentType` set to `id-ITS-ISE-ct-EnrolmentRequest`
    - the `signedContent` containing the `InnerECRequest`
    - the `signedAttributes` collection containing an `attr-signingTime` attribute
    - the signer declared as self
    - the signature computed using the canonical private key
- an `EncryptedData` structure is built, with:
    - the recipients is the EA, the recipient public key to use is the `encryption_key` of the EA
    - the `encryptedContentType` set to `id-ITS-ISE-ct-SignedData`
    - the `encryptedContent` containing the encrypted representation of the `SignedData` structure
- a Data structure is built, with:
    - the `contentType` set to `id-ITS-ISE-ct-EncryptedData`
    - the content containing the `EncryptedData` structure

```
InnerECRequest ::= SEQUENCE {
requestIdentifier OCTET STRING (SIZE(16)),
itsId IA5String,
wantedSubjectAttributes SubjectAttributes,
wantedValidityRestrictions ValidityRestrictions OPTIONAL,
responseEncryptionKey PublicKey }
```

`wantedSubjectAttributes` is the serialization of the subject_attributes structure defined in ETSI Standard [1]; it must contain exactly one instance of the following elements:

- a `verification_key`,
- an `its_aid_ssp_list`

`wantedValidityRestrictions` is the serialization of the `subject_validity_restrictions` defined in ETSI Standard [1]; this field is optional because the EA already knows the ITS-S and can set duration and region restrictions on its own.

The `requestIdentifier` can be reused by the ITS-S if network connectivity has been lost during the transaction. In that case, it is expected to send the exact same request.

Security characteristics

- Identity is ensured by the `itsId` present in the request.
- Integrity is ensured by the signature and verified by checking the signature against the canonical public key associated to this `itsId`.
- Confidentiality is ensured by encrypting the request with the encryption key of the EA.
- Anonymity of the requestor toward an external attacker is ensured by the confidentiality of the request and its signature. Anonymity of the requestor toward the EA isn't a concern (EA must know and recognize the requestor).

3.3.3.2 Response format

The ITS-S shall receive a Data structure, containing an `EncryptedData` structure, containing a `SignedData` structure, containing an `InnerECResponse` structure. In some specific error cases, the `EncryptedData` structure can be missing, for example if the EA hasn't been able to read or validate the `responseEncryptionKey` in the request.

- if the EA has been able to read and validate the `responseEncryptionKey` in the request:
    - the outermost structure is a Data structure with its `contentType` set to `id-ITS-ISE-ct-EncryptedData`
    - the content octet string contains an `EncryptedData` structure, with:
        - recipients references the `responseEncryptionKey` set in the request, the recipient identifier is computed as described in section `EncryptedData`
        - the `encryptedContentType` is set to `id-ITS-ISE-ct-SignedData`
        - the `encryptedContent`, once decrypted, contains a `SignedData` structure
- if the EA hasn't been able to read and validate the `responseEncryptionKey` in the request:
    - the outermost structure is a Data structure with its `contentType` set to `id-ITS-ISE-ct-SignedData`
    - the content contains a `SignedData` structure

In both cases, this expected SignedData structure is:

- the `signedContentType` is set to `id-ITS-ISE-ct-EnrolmentResponse`
- the `signedContent` contains the `InnerECResponse`
- the `signer` is populated with the `certificateDigest` field, containing the HashedId8 of the EA
- the signature is computed using the EA private key corresponding to its public `verification_key` found in the EA certificate

The InnerECResponse shall contain:

- the r`equestHash` is the left-most 16 octets of the SHA256 digest of the Data structure received in the request
- a `responseCode` indicating the result of the request
- if `responseCode` is 0, indicating a positive response, then a certificate is returned, and optionally a CA contribution value for the ITS to compute its private key (implicit certificates using ECQV)
- if `responseCode` is different than 0, indicating a negative response, then no certificate and no CA contribution value will be returned

```
InnerECResponse ::= SEQUENCE {
    requestHash OCTET STRING (SIZE(16)),
    responseCode EnrolmentResponseCode,
    certificate Certificate OPTIONAL,
    cAContributionValue INTEGER OPTIONAL }
-- requestHash is a truncated SHA256 of the whole Data structure received
```

```
EnrolmentResponseCode ::= ENUMERATED {
    ok(0),
    cantparse, -- valid for any structure
    badcontenttype, -- not encrypted, not signed, not enrolmentrequest
    imnottherecipient, -- the "recipients" doesn't include me
    unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
    decryptionfailed, -- works for ECIES-HMAC and AES-CCM
    unknownits, -- can't retrieve the ITS from the itsId
    invalidsignature, -- signature verification of the request fails
    invalidencryptionkey, -- signature is good, but the responseEncryptionKey is bad
    baditsstatus, -- revoked, not yet active
```

```
            incompleterequest, -- some elements are missing
            deniedpermissions, -- requested permissions are not granted
            invalidkeys, -- either the verification_key of the encryption_key is bad
            deniedrequest, -- any other reason?
            ... }
```

Security characteristics

- Identity is ensured by the signer identifier of the `SignedData` structure (contains the HashedId8 of the EA).
- Integrity is ensured by the signature and verified by checking the signature against the `verification_key` of the EA.
- Confidentiality is ensured by encrypting the response to the `responseEncryptionKey` provided in the request. If this key wasn't valid, confidentiality isn't ensured, but no personal information is returned.
- Anonymity of the requestor toward an external attacker is ensured by the absence of identifiable information returned when no encryption is possible, and by encryption of the response where possible.

3.3.4 Request Authorization Ticket (AT)

POST http://aa_access_point

Inputs:

• Content-type: application/x-its-request

• Content: binary encoded `AuthorizationRequest` object

Outputs:

• Content-type: application/x-its-response

• Content: binary encoded `AuthorizationResponse` object

3.3.4.1 Request format

The ITS-S must build its AT request by following this process:

- an ECC private key is randomly generated (the response-decryption-key), the corresponding public key is computed (response-encryption-key)
- a random 32 octets long secret key (`hmac-key`) is generated
- a tag using the HMAC-SHA256 function is computed using the previously generated `hmac-key`, on the concatenation of the serialization of `verificationKey` and `encryptionKey` elements (`encryptionKey` is optional); this tag is truncated to the leftmost 128 bits and named `keyTag` (see [11]).
- a `SharedATRequest` structure is built, with:
    - a randomly generated `requestIdentifier`
    - the `eaId` identifying the EA to contact for verification
    - the calculated `keyTag`
    - the desired attributes
    - some optional restrictions
    - a desired start date and time
    - the `response-encryption-key`
- a `SignedData` structure is built, with:
    - the `signedContentType` set to `id-ITS-ISE-ct-SharedATRequest`
    - the `signedAttributes` collection containing an `attr-signingTime` attribute
    - the `signedContent` is absent (external signature)

    - the `signer` declared as a `certificateDigest` referencing the EC
    - the signature computed using the EC verification private key
- an `EncryptedData` structure is built, with:
    - the `recipients` is the EA, the recipient public key to use is the `encryption_key` of the EA
    - the `encryptedContentType` set to `id-ITS-ISE-ct-SignedData`
    - the `encryptedContent` containing the encrypted representation of the previous `SignedData` structure
- an `InnerATRequest` structure is built, containing:
    - the `verificationKey requested` for certification
    - an optional `encryptionKey` to be placed in the same certificate
    - the `generated hmac-key`
    - the `signedByEC` containing the `SharedATRequest` structure
    - the `detachedEncryptedSignature` containing the previous `EncryptedData` structure
- an `EncryptedData` structure is built, with:
    - the `recipients` is the AA, the recipient public key to use is the `encryption_key` of the AA
    - the `encryptedContentType` set to `id-ITS-ISE-ct-AuthorizationRequest`
    - the `encryptedContent` containing the encrypted representation of the `InnerATRequest` structure
- a `Data` structure is built, with:
    - the `contentType` set to `id-ITS-ISE-ct-EncryptedData`
    - the `content` containing the previous `EncryptedData` structure

`wantedSubjectAttributes` shall not contain a `verification_key` or an `encryption_key attribute`, but shall contain an `its_aid_ssp_list attribute`.

```
        SharedATRequest ::= SEQUENCE {
            requestIdentifier OCTET STRING (SIZE(16)),
            eaId HashedId8,
            keyTag OCTET STRING (SIZE(16)),
            wantedSubjectAttributes SubjectAttributes,
            wantedValidityRestrictions ValidityRestrictions OPTIONAL,
            wantedStart Time32,
            responseEncryptionKey PublicKey }
```

```
        InnerATRequest ::= SEQUENCE {
            verificationKey PublicKey,
            encryptionKey PublicKey OPTIONAL,
            hmacKey OCTET STRING (SIZE(32)),
            signedByEC SharedATRequest,
            detachedEncryptedSignature EncryptedData }
```

The figure 4 illustrates the structure of AT request.

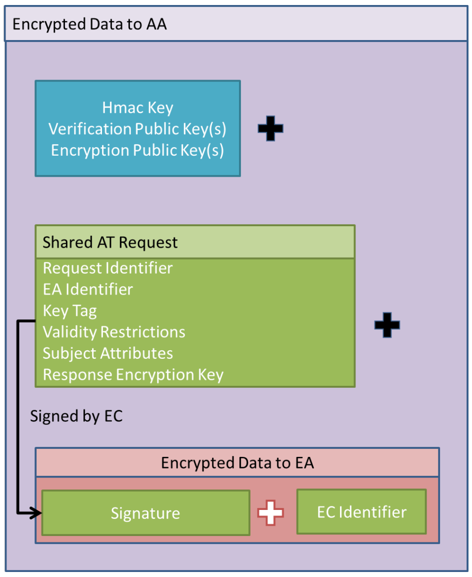

Figure 4

Security characteristics

- Identity is ensured by the signer identifier present in the encrypted signature.
- Integrity is ensured by the signature and verified by checking the signature against the public key associated to this signer (found in the corresponding EC). The signature indirectly covers the `verificationKey` and `encryptionKey` elements, by their digests (second pre-image resistance of the hash function, which is greater than the collision resistance used in signatures). The AA cannot verify the signature, only the EA can do it, but the AA can verify the requested permissions, and can verify that the HMAC signature of the public keys match the given `keyTag`.
- Confidentiality toward an external attacker is ensured by encrypting the request to the encryption key of the AA.
- Anonymity of the requestor toward an external attacker is ensured by the confidentiality of the request and its signature. Anonymity of the requestor toward the AA is ensured by the additional encryption of the signature and the signer. Anonymity of the requestor toward the EA isn't a concern (the EA must know and recognize the requestor).
- Unlinkability of the authorization tickets toward an external attacker is ensured by the confidentiality characteristics. Unlinkability of the authorization tickets toward the AA is ensured by the additional encryption of the signature and the signer. Unlinkability of the authorization tickets toward the EA is ensured by hiding the final public keys to certify from the EA.

3.3.4.2 Response format

The ITS-S shall receive a Data structure, containing an `EncryptedData` structure, containing a `SignedData` structure, containing an `InnerATResponse` structure. In some specific error cases, the `EncryptedData` structure can be missing, for example if the AA hasn't been able to read or validate the `responseEncryptionKey` in the request.

- if the AA has been able to read and validate the `responseEncryptionKey` in the request:
  - the outermost structure is a Data structure with its `contentType` set to `id-ITS-ISE-ct-EncryptedData`
  - the content octet string contains an `EncryptedData` structure, with:
    - recipients references the `responseEncryptionKey` set in the request, the recipient identifier is computed as described in section `EncryptedData`

        - the `encryptedContentType` is set to `id-ITS-ISE-ct-SignedData`
        - the `encryptedContent`, once decrypted, contains a `SignedData` structure
- if the AA hasn't been able to read and validate the `responseEncryptionKey` in the request:
    - the outermost structure is a Data structure with its `contentType` set to `id-ITS-ISE-ct-SignedData`
    - the content contains a `SignedData` structure

In both cases, this expected `SignedData` structure is:

- the `signedContentType` is set to `id-ITS-ISE-ct-AuthorizationResponse`
- the `signedContent` contains the `InnerATResponse`
- the signer is populated with the `certificateDigest` field, containing the HashedId8 of the AA
- the signature is computed using the AA private key corresponding to its public `verification_key` found in the AA certificate

The `InnerATResponse` shall contain:

- the `requestHash` is the left-most 16 octets of the SHA256 digest of the Data structure received in the request
- a `responseCode` indicating the result of the request
- if `responseCode` is 0, indicating a positive response, then a certificate is returned, and optionally a CA contribution value for the ITS to compute its private key (implicit certificates using ECQV)
- if `responseCode` is different than 0, indicating a negative response, then no certificate and no CA contribution value will be returned

```
        InnerATResponse ::= SEQUENCE {
            requestHash OCTET STRING (SIZE(16)),
            responseCode AuthorizationResponseCode,
            certificate Certificate OPTIONAL,
            cAContributionValue INTEGER OPTIONAL }
        -- requestHash is a truncated SHA256 of the whole Data structure received
```

```
        AuthorizationResponseCode ::= ENUMERATED {
            ok(0), -- ITS->AA
            its-aa-cantparse, -- valid for any structure
            its-aa-badcontenttype, -- not encrypted, not signed, not authorizationrequest
            its-aa-imnottherecipient, -- the "recipients" doesn't include me
            its-aa-unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
            its-aa-decryptionfailed, -- works for ECIES-HMAC and AES-CCM
            its-aa-keysdontmatch, -- HMAC keyTag verification fails
            its-aa-incompleterequest, -- some elements are missing
            its-aa-invalidencryptionkey, -- the responseEncryptionKey is bad
            its-aa-outofsyncrequest, -- signingTime is outside acceptable limits
            its-aa-unknownea, -- the EA identified by eaId is unknown to me
            its-aa-invalidea, -- the EA certificate is revoked
            its-aa-deniedpermissions, -- I, the AA, deny the requested permissions  -- AA->EA
            aa-ea-cantreachea, -- the EA is unreachable (network error?)    -- EA->AA
            ea-aa-cantparse, -- valid for any structure
            ea-aa-badcontenttype, -- not encrypted, not signed, not authorizationrequest
            ea-aa-imnottherecipient, -- the "recipients" of the outermost encrypted data
doesn't include me
            ea-aa-unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
            ea-aa-decryptionfailed, -- works for ECIES-HMAC and AES-CCM  -- TODO: continuer
            invalidaa, -- the AA certificate presented is invalid/revoked/whatever
            invalidaasignature, -- the AA certificate presented can't validate the request
signature
            wrongea, -- the encrypted signature doesn't designate me as the EA
            unknownits, -- can't retrieve the EC/ITS in my DB
            invalidsignature, -- signature verification of the request by the EC fails
```

```
           invalidencryptionkey, -- signature is good, but the key is bad
           deniedpermissions, -- permissions not granted
           deniedtoomanycerts, -- parallel limit
           ... }
```

Security characteristics

- Identity is ensured by the signer identifier of the `SignedData` structure (contains the HashedId8 of the AA).
- Integrity is ensured by the signature and verified by checking the signature against the `verification_key` of the AA.
- Confidentiality is ensured by encrypting the response to the `responseEncryptionKey` provided in the request. If this key wasn't valid, confidentiality isn't ensured, but no personal information is returned.
- Anonymity of the requestor toward an external attacker is ensured by the absence of identifiable information returned when no encryption is possible, and by encryption of the response where possible.

3.3.5 Validate Authorization Ticket (AT) request

POST http://ea_access_point

Inputs:

• Content-type: application/x-its-request

• Content: binary encoded `AuthorizationValidationRequest` object

Outputs:

• Content-type: application/x-its-response

• Content: binary encoded `AuthorizationValidationResponse` object

3.3.5.1 Request format

The AA must build its permissions verification request by following this process:

- an ECC private key is randomly generated (the response-decryption-key), the corresponding public key is computed (response-encryption-key)
- an `AuthorizationValidationRequest` structure is built, with:
  - a randomly generated `requestIdentifier`
  - the `sharedATRequest` containing the `signedByEC` submitted in the pseudonym certificate request
  - the `detachedEncryptedSignature` submitted in the same pseudonym certificate request
  - the response-encryption-key
- a `SignedData` structure is built, with:
  - the `signedContentType` set to `id-ITS-ISE-ct-AuthorizationValidationRequest`
  - the `signedContent` containing the `AuthorizationValidationRequest`
  - the `signedAttributes` collection containing an `attr-signingTime` attribute
  - the signer declared as certificate and contains the AA certificate
  - the signature computed using the AA signature private key
- an `EncryptedData` structure is built, with:
  - the recipients is the EA, the recipient public key to use is the `encryption_key` of the EA
  - the `encryptedContentType` set to `id-ITS-ISE-ct-SignedData`
  - the `encryptedContent` containing the encrypted representation of the `SignedData` structure
- a Data structure is built, with:

- the `contentType` set to `id-ITS-ISE-ct-EncryptedData`
- the content containing the `EncryptedData` structure

```
        AuthorizationValidationRequest ::= SEQUENCE {
            requestIdentifier OCTET STRING (SIZE(16)),
            sharedATRequest SharedATRequest,
            detachedEncryptedSignature EncryptedData,
            responseEncryptionKey PublicKey }
```

The figure 5 illustrates the structure of authorization validation request.

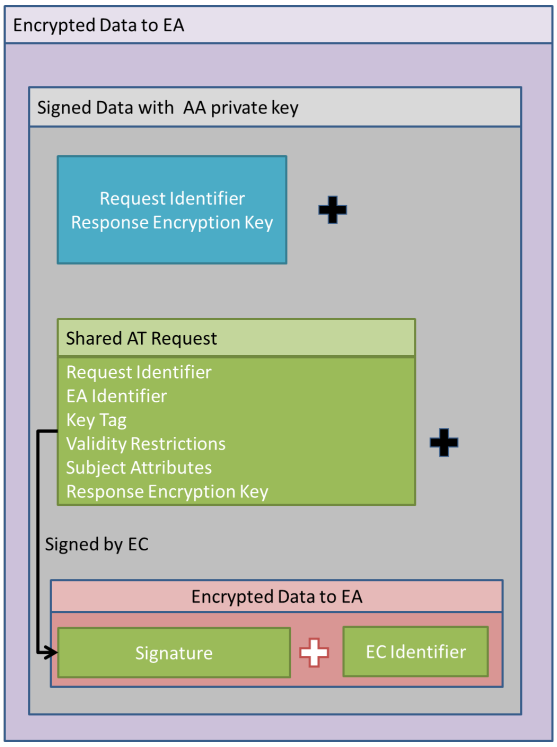

Figure 5

Security characteristics

- Identity is ensured by the AA certificate used as the signer identifier in the `SignerInfo` structure.
- Integrity is ensured by the signature and verified by checking the signature against the verification public key assessed in this certificate. The validity of the requestor AA is verified by chaining the certificate to a trusted root.
- Confidentiality is ensured by encrypting the request to the encryption key of the EA.
- Anonymity of the ITS-S toward an external attacker is ensured by the confidentiality of the request.

3.3.5.2 Response format

The AA shall receive a Data structure, containing an `EncryptedData` structure, containing a `SignedData` structure, containing an `AuthorizationValidationResponse` structure. In some specific error cases, the `EncryptedData` structure can be missing, for example if the EA hasn't been able to read or validate the `responseEncryptionKey` in the request.

- if the EA has been able to read and validate the `responseEncryptionKey` in the request:

  - the outermost structure is a Data structure with its `contentType` set to `id-ITS-ISE-ct-EncryptedData`
  - the content octet string contains an `EncryptedData` structure, with:
    - recipients references the `responseEncryptionKey` set in the request, the recipient identifier is computed as described in section `EncryptedData`
    - the `encryptedContentType` is set to `id-ITS-ISE-ct-SignedData`
    - the `encryptedContent`, once decrypted, contains a `SignedData` structure
- if the EA hasn't been able to read and validate the `responseEncryptionKey` in the request:
  - the outermost structure is a Data structure with its `contentType` set to `id-ITS-ISE-ct-SignedData`
  - the content contains a `SignedData` structure

In both cases, this expected `SignedData` structure is:

- the `signedContentType` is set to `id-ITS-ISE-ct-AuthorizationValidationResponse`
- the `signedContent` contains the `AuthorizationValidationResponse`
- the signer is populated with the `certificateDigest` field, containing the HashedId8 of the EA
- the signature is computed using the EA private key corresponding to its public `verification_key` found in the EA certificate

The `AuthorizationValidationResponse` shall contain:

- the `requestHash` is the left-most 16 octets of the SHA256 digest of the Data structure received in the request
- a `responseCode` indicating the result of the request
- if `responseCode` is 0, indicating a positive response, then `subjectAssurance`, `startDate` and `endDate` are returned to be set in corresponding AT
- if `responseCode` is different than 0, indicating a negative response, then no `subjectAssurance`, no `startDate`, and no `endDate` are returned

```
       AuthorizationValidationResponse ::= SEQUENCE {
           requestHash OCTET STRING (SIZE(16)),
           responseCode AuthorizationValidationResponseCode,
           subjectAssurance SubjectAssurance OPTIONAL,
           startDate [0] Time32 OPTIONAL,
           endDate [1] Time32 OPTIONAL }

       -- requestHash is a truncated SHA256 of the whole Data structure received
```

```
       AuthorizationValidationResponseCode ::= ENUMERATED {
           ok(0),
           cantparse, -- valid for any structure
           badcontenttype, -- not encrypted, not signed, not permissionsverificationrequest
           imnottherecipient, -- the "recipients" of the outermost encrypted data doesn't
include me
           unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
           decryptionfailed, -- works for ECIES-HMAC and AES-CCM
           invalidaa, -- the AA certificate presented is invalid/revoked/whatever
           invalidaasignature, -- the AA certificate presented can't validate the request
signature
           wrongea, -- the encrypted signature doesn't designate me as the EA
           unknownits, -- can't retrieve the EC/ITS in my DB
           invalidsignature, -- signature verification of the request by the EC fails
           invalidencryptionkey, -- signature is good, but the responseEncryptionKey is bad
           deniedpermissions, -- requested permissions not granted
           deniedtoomanycerts, -- parallel limit
           deniedrequest, -- any other reason?
           ... }
```

Security characteristics

- Identity is ensured by the signer identifier of the `SignedData` structure (contains the HashedId8 of the EA).
- Integrity is ensured by the signature and verified by checking the signature against the `verification_key` of the EA.
- Confidentiality is ensured by encrypting the response to the `responseEncryptionKey` provided in the request. If this key wasn't valid, confidentiality isn't ensured, but no personal information is returned.
- Anonymity of the ITS-S requesting a pseudonym certificate toward an external attacker is ensured by the absence of identifiable information returned when no encryption is possible, and by encryption of the response where possible.

3.3.6 Get CRL

GET http://dc_access_point/getcrl/HashedId8

The abs_path part of the HTTP request is built by taking the DC access point (from the TSL or from an ad-hoc configuration), appending "/getcrl/", and the uppercase hexadecimal representation of HashedId8.

Inputs:

• No inputs

Outputs:

• Content-type: application/x-its-crl

• Content: binary encoded CRL object issued by the entity identified byHashedId8

The format of CRL is described in section 3.2.6.

3.3.7 Get TSL

GET http://dc_access_point/gettsl/HashedId8

The abs_path part of the HTTP request is built by taking the DC access point (from the TSL or from an ad-hoc configuration), appending "/gettsl/", and the uppercase hexadecimal representation of HashedId8.

Inputs:

• No inputs

Outputs:

• Content-type: application/x-its-tsl

• Content: binary encoded TSL object issued by the entity identified by HashedId8

The format of TSL is described in section 3.2.7.

Appendix A: Examples of request

1. <u>Enrolment Credential request example</u>

The ITS-S whose canonical ID "Renault-123456" requests an EC usable for CAM and DENM with some permission, and no validity restriction. The `InnerECRequest` content is:

```
      innerecreq InnerECRequest ::= {
        requestIdentifier 'E665759B9756D789FCCB1B2577E46A66'H,
        itsId "Renault-123456",
        wantedSubjectAttributes '30
000002D50E7A16DEF1F5E2FB22F85ED8FC4E9F8D22404061EE6F22290280807CC223F2
2109240301000025040100000'H, -- a verification_key and 2 ITSAIDSSP (CAM&DENM)
        responseEncryptionKey {
          type compressed-lsb-y-0,
          x '77BCBC87A68ECFE8CD7DD6CDC0320A9806996CF5A08D72C3226450E68BF33BD0'H
        }
      }
```

The DER encoding of this `innerecreq` is the following octet stream, 126 octets long, here beautified for readability:

```
      30 7C -- InnerECRequest
        04 10 E665759B9756D789FCCB1B2577E46A66 -- requestIdentifier
        16 0E 52656E61756C742D313233343536 -- itsId
        04 31
30000002D50E7A16DEF1F5E2FB22F85ED8FC4E9F8D22404061EE6F22290280807CC223F221092403010000250401
000000 -- wantedSubjectAttributes
        30 25 -- responseEncryptionKey
          0A 01 02 -- type
          02 20 77BCBC87A68ECFE8CD7DD6CDC0320A9806996CF5A08D72C3226450E68BF33BD0 -- x
```

This PDU is then encapsulated in a SignedData structure:

```
      signedreq SignedData ::= {
        version v1,
        hashAlgorithms {
          { algorithm id-sha256 }
        },
        signedContentType id-ITS-ISE-ct-EnrolmentRequest,
        signedContent '... here goes the innerecreq ...'H,
        signerInfos {
          {
            version v1,
            signer self:NULL,
            digestAlgorithm { algorithm id-sha256 },
            signatureAlgorithm { algorithm ecdsa-with-SHA256 },
            signedAttributes {
              {
                attrType id-messageDigest,
                attrValue OCTET STRING ::=
'AA349D9F1817AF5C662B04250427B3E2D07A027FD8AEA70114783661EA5DB11D'H -- SHA256 digest value
of innerecreq
              },
              {
                attrType id-contentType,
                attrValue OBJECT IDENTIFIER ::= id-ITS-ISE-ct-EnrolmentRequest
              },
              {
                attrType id-signingTime,
                attrValue INTEGER ::= 1426674524 -- 18 march 2015 10:28:44 UTC
              }
            },
            -- no certificateChain
          signature
'304502206982D1E49CA00BCE5F9DB81FDFEC06FE3AAC4915394FA7F171AED076E443C655022100DF88B8C08F5FA
3B57DEA4D66A5DBEDEF378CC7500D9F2DC13AC50BA0DAADCF10'H
         }
       }
     }
```

The DER encoding of this `signedreq` is the following octet stream, 344 octets long:

```
      30 82 0154 -- SignedData
        30 0D -- hashAlgorithms
          30 0B -- HashAlgorithmIdentifier
            06 09 608648016503040201 -- id-sha256
        06 0C 2B0601040181AD5A04010104 -- id-ITS-ISE-ct-EnrolmentRequest
        04 7E <...insert here the innerecreq...>
        30 81 B4 -- SignerInfos
          30 81 B1 -- SignerInfo
            30 66 -- signedAttributes
              30 30
                06 0C 2B0601040181AD5A04010301 -- id-messageDigest
                04 20 AA349D9F1817AF5C662B04250427B3E2D07A027FD8AEA70114783661EA5DB11D
              30 1C
                06 0C 2B0601040181AD5A04010302 -- id-ContentType
                06 0C 2B0601040181AD5A04010104
              30 14
                06 0C 2B0601040181AD5A04010303 -- id-signingTime
                02 04 5509535C
            04 47
304502206982D1E49CA00BCE5F9DB81FDFEC06FE3AAC4915394FA7F171AED076E443C655022100DF88B8C08F5FA3
B57DEA4D66A5DBEDEF378CC7500D9F2DC13AC50BA0DAADCF10 -- signature
```

This PDU is then encrypted using the AES-128-CCM mechanism with default ETSI Standard [1] parameters (this produces a 360 octets long octet string), and the AES key is encrypted using ECIES mechanism with default ETSI Standard [1] parameters to the EA identified by its HashedId8='0001020304050607'H. The resulting EncryptedData structure is built like this:

```
      encryptedreq EncryptedData ::= {
        version v1,
        recipients {
          {
            recipient '0001020304050607'H,
            kexalgid { algorithm id-ecies-103097 },
            encryptedKeyMaterial
'304C30260A0103022100ABC4563E98E4395FC2D968E2ADA4A310D49D5D9E4C929EC1F5EDF13F6D8797CC04107F6
4B447AF6913833C1C5F5BF60131930410E93749FF54892F24533A1EE746EF23C2'H -- contains an
ECIESEncryptedKey103097
          }
        },
        encryptedContentType id-ITS-ISE-ct-SignedData,
        contentEncryptionAlgorithm {
          algorithm aes-128-ccm-103097,
          parameters { aes-nonce '000102030405060708090A0B0C'H }
        },
        encryptedContent '... here goes the encrypted signedreq ...'H
      }
```

The DER encoding of this encryptedreq is the following octet stream, 507 octets long:

```
      30 82 01F7 -- EncryptedData
        30 5C -- recipients
          30 5A -- RecipientInfo
            04 08 0001020304050607 --recipient
            04 4E
304C30260A0103022100ABC4563E98E4395FC2D968E2ADA4A310D49D5D9E4C929EC1F5EDF13F6D8797CC04107F64
B447AF6913833C1C5F5BF60131930410E93749FF54892F24533A1EE746EF23C2 -- encryptedKeyMaterial
        06 0C 2B0601040181AD5A04010102 -- id-ITS-ISE-ct-SignedData
        30 1D -- encryptionAlgorithm
```

```
          06 0C 2B0601040181AD5A04010201 -- ce-aes-128-ccm-103097
          04 0D 000102030405060708090A0B0C -- aes-nonce
        04 82 0168 <...insert here the encrypted signedreq...>
```

This PDU is then encapsulated in a Data structure, built like this:

```
      enrolmentrequest Data ::= {
        version v1,
        contentType id-ITS-ISE-ct-EncryptedData,
        content '... here goes the encryptedrec ...'H
      }
```

The DER encoding of this `enrolmentrequest` is the following octet stream, 529 octets long:

```
      30 82 020B -- Data
        06 0C 2B0601040181AD5A04010103 -- id-ITS-ISE-ct-EncryptedData
        04 82 01FB <...insert here the encryptedreq...>
```

2. <u>Authorization Ticket request example</u>

An ITS-S requests an AT usable for CAM and DENM with some permission, no encryption key, and no validity restrictions. First, a `SharedATRequest` is built:

```
      sharedatreq SharedATRequest ::= {
        requestIdentifier '41E33B6C090187D2BAE0A4E8C5A77DC4'H,
        eaId '0001020304050607'H, -- the EA
        keyTag 'FA5BECEAA0E6E5B6088DE52EDAD6F18F'H,
        wantedSubjectAttributes '0D21092403010000250401000000'H, -- 2 ITSAIDSSP (CAM&DENM)
        -- no wantedValidityRestrictions
        wantedStart 1426723200, -- 19 march 2015 00:00:00 UTC
        responseEncryptionKey {
          type compressed-lsb-y-1,
          x 'F302F81307B7CA056023EA959EAB932D043AA7C86ACA6B4ECE8E8F5FDC35AE4F'H
        }
      }
```

The DER encoding of this `sharedatreq` is the following octet stream, 110 octets long:

```
      30 6C -- SharedATRequest
        04 10 41E33B6C090187D2BAE0A4E8C5A77DC4 -- requestIdentifier
        04 08 0001020304050607 -- eaId
        04 10 FA5BECEAA0E6E5B6088DE52EDAD6F18F -- keyTag
        04 0E 0D21092403010000250401000000 -- wantedSubjectAttributes
        02 04 550A1180 -- wantedStart
        30 26 -- responseEncryptionKey
          0A 01 03 -- type
          02 21 00F302F81307B7CA056023EA959EAB932D043AA7C86ACA6B4ECE8E8F5FDC35AE4F -- x
```

This `sharedatreq` needs to be signed, so a SignedData structure is built:

```
      signedextsharedatreq SignedData ::= {
        version v1,
        hashAlgorithms {
          { algorithm id-sha256 }
        },
        signedContentType id-ITS-ISE-ct-SharedATRequest,
        -- no signedContent, this is an external signature
        signerInfos {
          {
            version v1,
            signer certificateDigest {
              algorithm { algorithm id-sha256 },
              digest '97583D6CE5C46B5E'H -- this is the HashedId8 of the EC
            },
            digestAlgorithm { algorithm id-sha256 },
            signatureAlgorithm { algorithm ecdsa-with-SHA256 },
            signedAttributes {
              {
                 attrType id-messageDigest,
                 attrValue OCTET STRING ::=
'01E10ED2BD3E0FFB451FD64036ED12A1B5942F78365CF39D5F22C9A3DF3F697A'H -- SHA256 digest value
of sharedatreq
               },
               {
                 attrType id-contentType,
                 attrValue OBJECT IDENTIFIER ::= id-ITS-ISE-ct-SharedATRequest
               },
               {
                 attrType id-signingTime,
                 attrValue INTEGER ::= 1426674528 -- 18 march 2015 10:28:48 UTC
               }
             },
             -- no certificateChain
             signature
'304402201C1B4CCA76525F1830A22E7E6B8F6ABEAABC72B0ECAC175CEF6601CA35726AFD02205931C93E92E0D58
BC6B43EBFE75F29B1BDD4289EBE8E3467F2D640F800CC6234'H
           }
         }
       }
```

The DER encoding of this `signedextsharedatreq` is the following octet stream, 226 octets long:

```
       30 81 DF -- SignedData
         30 0D -- hashAlgorithms
           30 0B -- HashAlgorithmIdentifier
             06 09 608648016503040201 -- id-sha256
         06 0C 2B0601040181AD5A0401010A
         30 81 BF -- signerInfos
           30 81 BC -- SignerInfo
             30 0A -- signer
               04 08 97583D6CE5C46B5E -- digest of EC
             30 66 -- signedAttributes
               30 30
                 06 0C 2B0601040181AD5A04010301 -- id-messageDigest
                 04 20 01E10ED2BD3E0FFB451FD64036ED12A1B5942F78365CF39D5F22C9A3DF3F697A
               30 1C
                 06 0C 2B0601040181AD5A04010302 -- id-ContentType
                 06 0C 2B0601040181AD5A0401010A
               30 14
                 06 0C 2B0601040181AD5A04010303 -- id-signingTime
                 02 04 55095360
             04 46
304402201C1B4CCA76525F1830A22E7E6B8F6ABEAABC72B0ECAC175CEF6601CA35726AFD02205931C93E92E0D58B
C6B43EBFE75F29B1BDD4289EBE8E3467F2D640F800CC6234 -- signature
```

This PDU is then encrypted using the AES-128-CCM mechanism with default ETSI Standard [1] parameters (this produces a 242 octets long octet string), and the AES key is encrypted using ECIES mechanism with default ETSI Standard [1] parameters to the EA identified by its HashedId8='0001020304050607'H. The resulting EncryptedData structure is built like this:

```
       encryptedsignedextsharedatreq EncryptedData ::= {
         version v1,
         recipients {
           {
             recipient '0001020304050607'H,
             kexalgid { algorithm id-ecies-103097 },
             encryptedKeyMaterial
'304C30260A01030221008FE956196A3F36BD514AD219CAC462DC13B1F99C98BEAF8CDE6C64269A55DA6C04108B5
B8E36EAB36577F0B76270C45D1D8204103E05A6E942F0BEE2A12779BEBA7577E1'H -- contains an
ECIESEncryptedKey103097
           }
         },
         encryptedContentType id-ITS-ISE-ct-SignedData,
         contentEncryptionAlgorithm {
           algorithm aes-128-ccm-103097,
           parameters { aes-nonce '000102030405060708090A0B0D'H }
         },
         encryptedContent '... here goes the encrypted signedextsharedatreq ...'H
       }
```

The DER encoding of this `encryptedsignedextsharedatreq` is the following octet stream, 372 octets long:

```
      30 82 0170 -- EncryptedData
        30 5C -- recipients
          30 5A -- RecipientInfo
            04 08 0001020304050607 --recipient
            04 4E
304C30260A01030221008FE956196A3F36BD514AD219CAC462DC13B1F99C98BEAF8CDE6C64269A55DA6C04108B5B
8E36EAB36577F0B76270C45D1D8204103E05A6E942F0BEE2A12779BEBA7577E1 -- encryptedKeyMaterial
        06 0C 2B0601040181AD5A04010102 -- id-ITS-ISE-ct-SignedData
        30 1D -- encryptionAlgorithm
          06 0C 2B0601040181AD5A04010201 -- ce-aes-128-ccm-103097
          04 0D 000102030405060708090A0B0D -- aes-nonce
        04 81 E2 <...insert here the encrypted signedextsharedatreq...>
```

The `sharedatreq`, the `encryptedsignedextsharedatreq`, public keys, and HMAC key are then encapsulated in an `InnerATRequest`:

```
      inneratreq InnerATRequest ::= {
        verificationKey {
          type compressed-lsb-y-1,
          x 'A009A3032AF6E9DC00BF70A9E36C84275A1CA8087A12245A7EB5DE2B2C805166'H
        },
        -- no encryptionKey
        hmacKey '60B316FD92AB81B793D5207F11AE34CF5AF6BA425A0B8395E2371DEB5479D3A2'H,
        signedByEC '... here goes the sharedatreq ...',
        detachedEncryptedSignature '... here goes the encryptedsignedextsharedatreq ...'
      }
```

The DER encoding of this `inneratreq` is the following octet stream, 560 octets long:

```
      30 82 022C -- InnerATRequest
        30 26 -- verificationKey
          0A 01 03 -- type
          02 21 A009A3032AF6E9DC00BF70A9E36C84275A1CA8087A12245A7EB5DE2B2C805166 -- x
        -- no encryptionKey
        04 20 60B316FD92AB81B793D5207F11AE34CF5AF6BA425A0B8395E2371DEB5479D3A2 -- hmacKey
        30 6C <...insert here the rest of the sharedatreq...>
        30 82 0170 <...insert here the rest of the encryptedsignedextsharedatreq...>
```

This PDU is then encrypted using the AES-128-CCM mechanism with default ETSI Standard [1]parameters (this produces a 576 octets long octet string), and the AES key is encrypted using ECIES mechanism with default ETSI Standard [1] parameters to the AA identified by its HashedId8='08090A0B0C0D0E0F'H. The resulting EncryptedData structure is built like this:

```
      encryptedreq EncryptedData ::= {
        version v1,
        recipients {
          {
            recipient '08090A0B0C0D0E0F'H,
            kexalgid { algorithm id-ecies-103097 },
            encryptedKeyMaterial
'304B30250A01030220214A61E116D709ABB38E211253A55BC66110C713C1253799AA1981A015A158060410E5A48
7625B458D28C96782E5FDB378A90410A3956CD0BA50F814F8BB6B6B4BCC5E1F'H -- contains an
ECIESEncryptedKey103097
          }
        },
        encryptedContentType id-ITS-ISE-ct-AuthorizationRequest,
        contentEncryptionAlgorithm {
          algorithm aes-128-ccm-103097,
          parameters { aes-nonce '000102030405060708090A0B0E'H }
```

```
         },
         encryptedContent '... here goes the encrypted inneratreq ...'H
       }
```

The DER encoding of this `encryptedreq` is the following octet stream, 722 octets long:

```
       30 82 02CE -- EncryptedData
         30 5B -- recipients
           30 59 -- RecipientInfo
             04 08 08090A0B0C0D0E0F --recipient
             04 4D
304B30250A01030220214A61E116D709ABB38E211253A55BC66110C713C1253799AA1981A015A158060410E5A487
625B458D28C96782E5FDB378A90410A3956CD0BA50F814F8BB6B6B4BCC5E1F -- encryptedKeyMaterial
         06 0C 2B0601040181AD5A04010106 -- id-ITS-ISE-ct-AuthorizationRequest
         30 1D -- encryptionAlgorithm
           06 0C 2B0601040181AD5A04010201 -- ce-aes-128-ccm-103097
           04 0D 000102030405060708090A0B0E -- aes-nonce
         04 82 0240 <...insert here the encrypted inneratreq...>
```

This PDU is then encapsulated in a Data structure, built like this:

```
       authorizationrequest Data ::= {
         version v1,
         contentType id-ITS-ISE-ct-EncryptedData,
         content '... here goes the encryptedrec ...'H
       }
```

The DER encoding of this `authorizationrequest` is the following octet stream, 744 octets long:

```
       30 82 02E4 -- Data
         06 0C 2B0601040181AD5A04010103 -- id-ITS-ISE-ct-EncryptedData
         04 82 02D2 <...insert here the encryptedreq...>
```

Appendix B: Encryption of a message

This appendix describes cryptographic operations to be implemented to encrypt a message (any) according to the mechanisms used in ETSI Standards [1]. Message encryption is used for example to communicate between ITS-S and the PKI (EA / AA), and between the AA and EA entities of the PKI.

Encrypt a message m (N octets) from a sender to a receiver.

Assuming an elliptic curve (p: curve prime, G: base point, q: base point order).

Sender only knows the (certified) encryption public key “Kb” of the receiver.

KDF (): SHA256(S || counter)...

E (a, b): a xor b

E-1(a, b): a xor b

MAC (km, m): HMAC (km, m)

||: concatenation

- Sender generates a random AES key A (128 bits, 16 octets)
- Sender chooses a nonce n, 12 octets
- Sender encrypts the message m with AES-CCM mode, the key A, and the nonce n. The output is the encrypted message M with an authentication tag (N+16 octets).
- Sender generates an ephemeral private key r in [1, q-1], and the associated public key v=r.G, 33 octets if compressed
- Sender derives a shared secret S from receiver encryption public key (Kb): S=Px, with (Px, Py)=r.Kb (verify that P != 0,if not, back to previous step)
- Sender then derives a set of keys ke and km with derivation algorithm: (ke || km)=KDF(S), ke is 16 octets long, km is 32 octets long
- Sender encrypts the AES key: c=E(ke, A), c is 16 octets long
- Sender produces a tag on the encrypted message: t=MAC(km, c), t is 16 octets long
- Sender transmits to the receiver a message C containing:
- The identifier for the recipient's certificate (cert_id), 8 octets
- The encrypted message M
- The encryption parameters (algorithm identifier aes_128_ccm, nonce n), 13 octets
- The ephemeral public key (v)
- The encrypted key (c) with the associated tag (t)
    - *8+N+16+13+33+16+16: 102+N octets, plus protocol overheads.*

- Receiver has its private key kb, and receives the message C.
- Receiver derives a shared secret S=Px, with (Px, Py)=kb.v
- Receiver derives (ke || km)=KDF(S)
- Receiver checks that the tag t verifies MAC(km, c), if not, receiver returns an error message

Appendix C: ASN.1 module

```
ISEEnrolmentProtocolv1
    { iso(1) identified-organization(3) dod(6) internet(1) private(4)
      enterprise(1) opentrust(22234) innovation(4) ise(1) modules(0)
      iseenrolmentprotocolv1(0) }

-- version BIT STRING { v1990(0), v1994(1), v1997(2) } ::= v1997

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All
-- The types and values defined in this module are exported for use
-- in the other ASN.1 modules.  Other applications may use them for
-- their own purposes.

IMPORTS
  -- RFC5084 Appendix
    aes, id-aes128-CCM, id-aes256-CCM, AES-CCM-ICVlen
      FROM CMS-AES-CCM-and-AES-GCM
        { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
          pkcs-9(9) smime(16) modules(0) cms-aes-ccm-and-gcm(32) }

  -- RFC5480
    ecdsa-with-SHA256, ecdsa-with-SHA384
      FROM PKIX1Algorithms2008
        { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0) 45 }

  -- RFC 4055 [RSAOAEP]
    id-sha256, id-sha384
      FROM PKIX1-PSS-OAEP-Algorithms
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-pkix1-rsa-pkalgs(33) } ;


/************
-- OIDs
************/
-- For the ISE project, lets allocate OIDs under the OpenTrust arc
id-OpenTrust OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1)
private(4) enterprise(1) opentrust(22234) }
id-OT-Innovation OBJECT IDENTIFIER ::= { id-OpenTrust 4 }
id-OT-Innovation-ISE OBJECT IDENTIFIER ::= { id-OT-Innovation 1 }
id-ITS-ISE-ct OBJECT IDENTIFIER ::= { id-OT-Innovation-ISE 1 }
id-ITS-ISE-ct-Data OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 1 }
id-ITS-ISE-ct-SignedData OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 2 }
id-ITS-ISE-ct-EncryptedData OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 3 }
id-ITS-ISE-ct-EnrolmentRequest OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 4 }
id-ITS-ISE-ct-EnrolmentResponse OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 5 }
id-ITS-ISE-ct-AuthorizationRequest OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 6 }
id-ITS-ISE-ct-AuthorizationResponse OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 7 }
id-ITS-ISE-ct-AuthorizationValidationRequest OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 8 }
id-ITS-ISE-ct-AuthorizationValidationResponse OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 9 }
id-ITS-ISE-ct-SharedATRequest OBJECT IDENTIFIER ::= { id-ITS-ISE-ct 10 }

id-ITS-ISE-algos OBJECT IDENTIFIER ::= { id-OT-Innovation-ISE 2 }
id-aes128-CCM-103097 OBJECT IDENTIFIER ::= { id-ITS-ISE-algos 1 }
id-ecies-103097 OBJECT IDENTIFIER ::= { id-ITS-ISE-algos 2 }

id-ITS-ISE-attrs OBJECT IDENTIFIER ::= { id-OT-Innovation-ISE 3 }
id-messageDigest OBJECT IDENTIFIER ::= { id-ITS-ISE-attrs 1 }
id-contentType OBJECT IDENTIFIER ::= { id-ITS-ISE-attrs 2 }
id-signingTime OBJECT IDENTIFIER ::= { id-ITS-ISE-attrs 3 }
```

```
-- From FIPS 202 draft
id-sha3-256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
    country(16) us(840) organization(1) gov(101)
    csor(3) nistalgorithm(4) hashalgs(2) 8 }


/************
-- Misc
************/
Version ::= INTEGER { v1(0), v2(1) }
HashedId8 ::= OCTET STRING (SIZE(8))
Time32 ::= INTEGER (0..4294967295)
SubjectAssurance ::= OCTET STRING (SIZE(1))
Certificate ::= OCTET STRING
SubjectAttributes ::= OCTET STRING
ValidityRestrictions ::= OCTET STRING
ContentType ::= OBJECT IDENTIFIER

PublicKey ::= SEQUENCE {
    type ECCPublicKeyType,
    x INTEGER }

ECCPublicKeyType ::= ENUMERATED {
    compressed-lsb-y-0(2),
    compressed-lsb-y-1(3) }

SignatureValue ::= OCTET STRING

-- SignatureValue should be opaque to the user/caller of security functions.
-- Internally, an ECDSA signature contains the following structure:
Ecdsa-Sig-Value ::= SEQUENCE {
    r INTEGER,
    s INTEGER }


/************
-- A generic class for an algorithm
************/
ALGORITHM ::= CLASS {
    &id OBJECT IDENTIFIER UNIQUE,
    &Type OPTIONAL
} WITH SYNTAX {
    ID &id
    [PARMS &Type]
}


/************
-- Signature algorithms declarations
************/
sign-ecdsa-with-sha256 ALGORITHM ::= {
    ID ecdsa-with-SHA256 }

sign-ecdsa-with-sha384 ALGORITHM ::= {
    ID ecdsa-with-SHA384 }

-- No OID defined yet
-- sign-ecdsa-with-sha3-256 ALGORITHM ::= {
--     ID ecdsa-with-SHA3-256 }

SignatureFunctions ALGORITHM ::=
    {   sign-ecdsa-with-sha256
      | sign-ecdsa-with-sha384
      -- | sign-ecdsa-with-sha3-256
      , ... }


/************
```

```
-- Content encryption algorithm declarations
************/
CCMDefaultParameters ::= SEQUENCE {
    aes-nonce OCTET STRING (SIZE(12)) }

ce-aes-128-ccm-103097 ALGORITHM ::= {
    ID id-aes128-CCM-103097
    PARMS CCMDefaultParameters }

CCMParameters ::= SEQUENCE {
    aes-nonce OCTET STRING (SIZE(7..13)),
    aes-ICVlen AES-CCM-ICVlen DEFAULT 12 }

ce-aes-128-ccm ALGORITHM ::= {
    ID id-aes128-CCM
    PARMS CCMParameters }

ce-aes-256-ccm ALGORITHM ::= {
    ID id-aes256-CCM
    PARMS CCMParameters }

DataEncryptionFunctions ALGORITHM ::=
    {   ce-aes-128-ccm-103097
      | ce-aes-128-ccm
      | ce-aes-256-ccm
      , ... }


/************
-- Key exchange algorithms declarations
************/
-- ECIESParameters ::= SEQUENCE {
--  kdf KeyDerivationFunction OPTIONAL,
--  sym SymmetricEncryption OPTIONAL,
--  mac MessageAuthenticationCode OPTIONAL }

-- ke-ecies ALGORITHM ::= {
--  ID ecies-specifiedParameters
--  PARMS ECIESParameters }

ECIESEncryptedKey103097 ::= SEQUENCE {
    v PublicKey,
    c OCTET STRING (SIZE(16)),
    t OCTET STRING (SIZE(16)) }

ke-ecies-103097 ALGORITHM ::= {
    ID id-ecies-103097 }

KeyEncryptionFunctions ALGORITHM ::=
    {   ke-ecies-103097
      -- | ke-ecies,
      , ... }


/************
-- Hash algorithms declarations
************/
hash-sha256 ALGORITHM ::= {
    ID id-sha256 }

hash-sha384 ALGORITHM ::= {
    ID id-sha384 }

hash-sha3-256 ALGORITHM ::= {
    ID id-sha3-256 }

HashFunctions ALGORITHM ::=
    {   hash-sha256
      | hash-sha384
      | hash-sha3-256
      , ... }


/************
-- AlgorithmIdentifiers using the preceding ObjectSets
************/
SignatureAlgorithmIdentifier ::= SEQUENCE {
    algorithm ALGORITHM.&id({SignatureFunctions}),
    parameters ALGORITHM.&Type({SignatureFunctions}{@algorithm}) OPTIONAL }

ContentEncryptionAlgorithmIdentifier ::= SEQUENCE {
    algorithm ALGORITHM.&id({DataEncryptionFunctions}),
    parameters ALGORITHM.&Type({DataEncryptionFunctions}{@algorithm}) OPTIONAL }

HashAlgorithmIdentifier ::= SEQUENCE {
    algorithm ALGORITHM.&id({HashFunctions}),
    parameters ALGORITHM.&Type({HashFunctions}{@algorithm}) OPTIONAL }

KeyEncryptionAlgorithmIdentifier ::= SEQUENCE {
    algorithm ALGORITHM.&id({KeyEncryptionFunctions}),
    parameters ALGORITHM.&Type({KeyEncryptionFunctions}{@algorithm}) OPTIONAL }


/************
-- Attributes
************/
ATTRIBUTE ::= CLASS {
    &id OBJECT IDENTIFIER UNIQUE,
    &Type OPTIONAL
} WITH SYNTAX {
    ID &id
    [VALUE &Type]
}

attr-messageDigest ATTRIBUTE ::= {
    ID id-messageDigest
    VALUE OCTET STRING }

attr-contentType ATTRIBUTE ::= {
    ID id-contentType
    VALUE ContentType }

attr-signingTime ATTRIBUTE ::= {
    ID id-signingTime
    VALUE Time32 }

SupportedAttributes ATTRIBUTE ::=
    {   attr-messageDigest
      | attr-contentType
      | attr-signingTime
      , ... }

Attribute ::= SEQUENCE {
    attrType ATTRIBUTE.&id({SupportedAttributes}),
    attrValue ATTRIBUTE.&Type({SupportedAttributes}{@attrType}) OPTIONAL }


/************
-- Data
************/
Data ::= SEQUENCE {
    version Version DEFAULT v1,
    contentType ContentType,
    content OCTET STRING OPTIONAL }
```

```
/************
-- SignedData
************/
SignedData ::= SEQUENCE {
    version Version DEFAULT v1,
    hashAlgorithms HashAlgorithmsIdentifiers,
    signedContentType ContentType,
    signedContent OCTET STRING OPTIONAL,
    signerInfos SignerInfos }

HashAlgorithmsIdentifiers ::= SEQUENCE OF HashAlgorithmIdentifier

SignerInfos ::= SEQUENCE OF SignerInfo

SignerInfo ::= SEQUENCE {
    version Version DEFAULT v1,
    signer [0] SignerIdentifier DEFAULT self:NULL,
    digestAlgorithm [1] HashAlgorithmIdentifier DEFAULT { algorithm id-sha256 },
    signatureAlgorithm [2] SignatureAlgorithmIdentifier DEFAULT { algorithm ecdsa-with-SHA256 },
    signedAttributes SignedAttributes,
    certificateChain SEQUENCE OF Certificate OPTIONAL,
    signature SignatureValue }

SignerIdentifier ::= CHOICE {
    self NULL,
    certificateDigest CertificateDigest,
    certificate Certificate }

CertificateDigest ::= SEQUENCE {
    algorithm HashAlgorithmIdentifier DEFAULT { algorithm id-sha256 },
    digest HashedId8 }

SignedAttributes ::= SEQUENCE OF Attribute


/************
-- EncryptedData
************/
EncryptedData ::= SEQUENCE {
    version Version DEFAULT v1,
    recipients RecipientInfos,
    encryptedContentType ContentType,
    encryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
    encryptedContent OCTET STRING OPTIONAL }

RecipientInfos ::= SEQUENCE SIZE (1..MAX) OF RecipientInfo

RecipientInfo ::= SEQUENCE {
    recipient HashedId8,
    kexalgid KeyEncryptionAlgorithmIdentifier DEFAULT { algorithm id-ecies-103097 },
    encryptedKeyMaterial OCTET STRING }


/************
-- EnrolmentRequest/Response
************/
InnerECRequest ::= SEQUENCE {
    requestIdentifier OCTET STRING (SIZE(16)),
    itsId IA5String,
    wantedSubjectAttributes SubjectAttributes,
    wantedValidityRestrictions ValidityRestrictions OPTIONAL,
    responseEncryptionKey PublicKey }

InnerECResponse ::= SEQUENCE {
    requestHash OCTET STRING (SIZE(16)),
    responseCode EnrolmentResponseCode,
    certificate Certificate OPTIONAL,
    cAContributionValue INTEGER OPTIONAL }
(   WITH COMPONENTS { responseCode (ok), certificate PRESENT }
  | WITH COMPONENTS { responseCode ALL EXCEPT (ok), certificate ABSENT, cAContributionValue
ABSENT }
)
-- requestHash is a truncated SHA256 of the whole Data structure received

EnrolmentResponseCode ::= ENUMERATED {
    ok(0),
    cantparse, -- valid for any structure
    badcontenttype, -- not encrypted, not signed, not enrolmentrequest
    imnottherecipient, -- the "recipients" doesn't include me
    unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
    decryptionfailed, -- works for ECIES-HMAC and AES-CCM
    unknownits, -- can't retrieve the ITS from the itsId
    invalidsignature, -- signature verification of the request fails
    invalidencryptionkey, -- signature is good, but the responseEncryptionKey is bad
    baditsstatus, -- revoked, not yet active
    incompleterequest, -- some elements are missing
    deniedpermissions, -- requested permissions are not granted
    invalidkeys, -- either the verification_key of the encryption_key is bad
    deniedrequest, -- any other reason?
    ... }


/************
-- AuthorizationRequest/Response
************/
SharedATRequest ::= SEQUENCE {
    requestIdentifier OCTET STRING (SIZE(16)),
    eaId HashedId8,
    keyTag OCTET STRING (SIZE(16)),
    wantedSubjectAttributes SubjectAttributes,
    wantedValidityRestrictions ValidityRestrictions OPTIONAL,
    wantedStart Time32,
    responseEncryptionKey PublicKey }

InnerATRequest ::= SEQUENCE {
    verificationKey PublicKey,
    encryptionKey PublicKey OPTIONAL,
    hmacKey OCTET STRING (SIZE(32)),
    signedByEC SharedATRequest,
    detachedEncryptedSignature EncryptedData }

InnerATResponse ::= SEQUENCE {
    requestHash OCTET STRING (SIZE(16)),
    responseCode AuthorizationResponseCode,
    certificate Certificate OPTIONAL,
    cAContributionValue INTEGER OPTIONAL }
(   WITH COMPONENTS { responseCode (ok), certificate PRESENT }
  | WITH COMPONENTS { responseCode ALL EXCEPT (ok), certificate ABSENT, cAContributionValue
ABSENT }
)
-- requestHash is a truncated SHA256 of the whole Data structure received

AuthorizationResponseCode ::= ENUMERATED {
    ok(0),
    -- ITS->AA
    its-aa-cantparse, -- valid for any structure
    its-aa-badcontenttype, -- not encrypted, not signed, not authorizationrequest
    its-aa-imnottherecipient, -- the "recipients" of the outermost encrypted data doesn't include
me
    its-aa-unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
    its-aa-decryptionfailed, -- works for ECIES-HMAC and AES-CCM
    its-aa-keysdontmatch, -- HMAC keyTag verification fails
    its-aa-incompleterequest, -- some elements are missing
    its-aa-invalidencryptionkey, -- the responseEncryptionKey is bad
    its-aa-outofsyncrequest, -- signingTime is outside acceptable limits
    its-aa-unknownea, -- the EA identified by eaId is unknown to me
    its-aa-invalidea, -- the EA certificate is revoked
    its-aa-deniedpermissions, -- I, the AA, deny the requested permissions
    -- AA->EA
    aa-ea-cantreachea, -- the EA is unreachable (network error?)
    -- EA->AA
    ea-aa-cantparse, -- valid for any structure
    ea-aa-badcontenttype, -- not encrypted, not signed, not authorizationrequest
    ea-aa-imnottherecipient, -- the "recipients" of the outermost encrypted data doesn't include
me
    ea-aa-unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
    ea-aa-decryptionfailed, -- works for ECIES-HMAC and AES-CCM
    -- TODO: continuer
    invalidaa, -- the AA certificate presented is invalid/revoked/whatever
    invalidaasignature, -- the AA certificate presented can't validate the request signature
    wrongea, -- the encrypted signature doesn't designate me as the EA
    unknownits, -- can't retrieve the EC/ITS in my DB
    invalidsignature, -- signature verification of the request by the EC fails
    invalidencryptionkey, -- signature is good, but the key is bad
    deniedpermissions, -- permissions not granted
    deniedtoomanycerts, -- parallel limit
    ... }


/************
-- AuthorizationValidationRequest/Response
************/
AuthorizationValidationRequest ::= SEQUENCE {
    requestIdentifier OCTET STRING (SIZE(16)),
    sharedATRequest SharedATRequest,
    detachedEncryptedSignature EncryptedData,
    responseEncryptionKey PublicKey }

AuthorizationValidationResponse ::= SEQUENCE {
    requestHash OCTET STRING (SIZE(16)),
    responseCode AuthorizationValidationResponseCode,
    subjectAssurance SubjectAssurance OPTIONAL,
    startDate [0] Time32 OPTIONAL,
    endDate [1] Time32 OPTIONAL }
(   WITH COMPONENTS { responseCode (ok), subjectAssurance PRESENT }
  | WITH COMPONENTS { responseCode ALL EXCEPT (ok), subjectAssurance ABSENT, startDate ABSENT,
endDate ABSENT }
)
-- requestHash is a truncated SHA256 of the whole Data structure received

AuthorizationValidationResponseCode ::= ENUMERATED {
    ok(0),
    cantparse, -- valid for any structure
    badcontenttype, -- not encrypted, not signed, not permissionsverificationrequest
    imnottherecipient, -- the "recipients" of the outermost encrypted data doesn't include me
    unknownencryptionalgorithm, -- either kexalg or contentencryptionalgorithm
    decryptionfailed, -- works for ECIES-HMAC and AES-CCM
    invalidaa, -- the AA certificate presented is invalid/revoked/whatever
    invalidaasignature, -- the AA certificate presented can't validate the request signature
    wrongea, -- the encrypted signature doesn't designate me as the EA
    unknownits, -- can't retrieve the EC/ITS in my DB
    invalidsignature, -- signature verification of the request by the EC fails
    invalidencryptionkey, -- signature is good, but the responseEncryptionKey is bad
    deniedpermissions, -- requested permissions not granted
    deniedtoomanycerts, -- parallel limit
    deniedrequest, -- any other reason?
    ... }


/************
-- Standalone certificate request (similar to PKCS#10)
************/
ITSCertificateRequest ::= SEQUENCE {
    itsCertReq ITSCertificateRequestContent,
    signatureAlgorithm SignatureAlgorithmIdentifier DEFAULT { algorithm ecdsa-with-SHA256 },
    signature SignatureValue }
```

```
ITSCertificateRequestContent ::= SEQUENCE {
    version Version DEFAULT v1,
    subjectName OCTET STRING (SIZE(0..32)),
    subjectAttributes SubjectAttributes,
    validityRestrictions ValidityRestrictions }


/************
-- CRL
************/
Crl ::= SEQUENCE {
    unsignedCrl ToBeSignedCrl,
    signatureAlgorithm SignatureAlgorithmIdentifier,
    signature SignatureValue }
-- signature is applied on unsignedCrl

ToBeSignedCrl ::= SEQUENCE {
    version Version,
    signer SignerIdentifier,
    thisUpdate Time32,
    nextUpdate Time32,
    entries SEQUENCE OF HashedId8 }


/************
-- TSL
************/
Tsl ::= SEQUENCE {
    unsignedTsl ToBeSignedTsl,
    signatureAlgorithm SignatureAlgorithmIdentifier,
    signature SignatureValue }
-- signature is applied on unsignedTsl

ToBeSignedTsl ::= SEQUENCE {
    version Version,
    signerInfo SignerIdentifier,
    notBefore Time32,
    notAfter Time32,
    trustServices SEQUENCE OF TrustService }

TrustService ::= SEQUENCE {
    serviceId TRUSTSERVICE.&id ({TrustServiceSet}),
    serviceValue TRUSTSERVICE.&Value ({TrustServiceSet}{@serviceId}) }

TrustServiceSet TRUSTSERVICE ::=
    {   ts-foreignRoot
      | ts-renewedRoot
      | ts-ea
      | ts-aa
      | ts-distributionCenter
      | ts-otherTslPointer
      , ... }

TRUSTSERVICE ::= CLASS {
    &id ServiceType UNIQUE,
    &Value }
WITH SYNTAX {
    SYNTAX &Value
    ID &id }

ts-foreignRoot TRUSTSERVICE ::= {
    SYNTAX Certificate
    ID foreignRoot }

ts-renewedRoot TRUSTSERVICE ::= {
    SYNTAX SEQUENCE {
        rootCertificate Certificate,
        linkRootCertificate Certificate }
    ID renewedRoot }
```

```
ts-ea TRUSTSERVICE ::= {
    SYNTAX SEQUENCE {
        certificate Certificate,
        linkedCertificate Certificate OPTIONAL,
        accessPoint IA5String }
    ID ea }

ts-aa TRUSTSERVICE ::= {
    SYNTAX SEQUENCE {
        certificate Certificate,
        accessPoint IA5String }
    ID aa }

ts-distributionCenter TRUSTSERVICE ::= {
    SYNTAX IA5String
    ID distributionCenter }

ts-otherTslPointer TRUSTSERVICE ::= {
    SYNTAX IA5String
    ID otherTslPointer }

ServiceType ::= ENUMERATED {
    foreignRoot,
    renewedRoot,
    ea,
    aa,
    distributionCenter,
    otherTslPointer,
    ... }


END -- of ISEEnrolmentProtocolv1
```