INTERNATIONAL STANDARD

ISO 24534-4

First edition
2010-07-15

Automatic vehicle and equipment identification — Electronic registration identification (ERI) for vehicles —

Part 4: Secure communications using asymmetrical techniques

Identification automatique des véhicules et des équipements — Identification d'enregistrement électronique (ERI) pour les véhicules —

Partie 4: Communications sûres utilisant des techniques asymétriques

Reference number
ISO 24534-4:2010(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 24534-4 was prepared by the European Committee for Standardization (CEN) Technical Committee CEN/TC 278, *Road transport and traffic telematics*, in collaboration with Technical Committee ISO/TC 204, *Intelligent transport systems*, in accordance with the Agreement on technical cooperation between ISO and CEN (Vienna Agreement).

This first edition of ISO 24534-4 cancels and replaces ISO/TS 24534-4:2008, which has been technically revised.

ISO 24534 consists of the following parts, under the general title *Automatic vehicle and equipment identification — Electronic registration identification (ERI) for vehicles*:

— *Part 1: Architecture*

— *Part 2: Operational requirements*

— *Part 3: Vehicle data*

— *Part 4: Secure communications using asymmetrical techniques*

— *Part 5: Secure communications using symmetrical techniques*

Introduction

A quickly emerging need has been identified with administrations to improve the unique identification of vehicles for a variety of services. Situations are already occurring where manufacturers intend to fit lifetime tags to vehicles. Various governments are considering the needs and benefits of electronic registration identification (ERI) as a legal proof of vehicle identity with potential mandatory uses. There is commercial and economic justification in respect of both tags and infrastructure that a standard enables an interoperable solution.

ERI is a means of uniquely identifying road vehicles. The application of ERI will offer significant benefits over existing techniques for vehicle identification. It will be a suitable tool for the future management and administration of traffic and transport, including applications in free-flow, multi-lane traffic conditions with the capability to support mobile transactions. ERI addresses the need of authorities and other road users for a trusted electronic identification, including roaming vehicles.

This part of ISO 24534 specifies the application layer interfaces for the exchange of data between an onboard component containing the ERI data and a reader or writer inside or outside the vehicle.

The exchanged identification data consists of a unique vehicle identifier and may also include data typically found in the vehicle's registration certificate. The authenticity of the exchanged vehicle data can be further enhanced by ensuring data has been obtained by request from a commissioned device, with the data electronically signed by the registration authority.

In order to facilitate (international) resales of vehicles, the ERI interface includes provisions for another accredited registration authority to take over the registration of a vehicle.

The ERI interface supports confidentiality measures to adhere to (inter)national privacy regulation and to prevent other misuse of electronic identification of vehicles. A registration authority may authorize other authorities to access the vehicle's data. A holder of a registration certificate may authorize an additional service provider to identify the vehicle when he/she wants commercial service.

However, it is perceived that different users may have different requirements for authentication and confidentiality. This International Standard therefore supports different levels of security with maximum compatibility. Much attention is given to the interoperability of the component containing the ERI data and readers of various levels of capability, e.g. the identification of a vehicle with a less capable ERI data component by a more sophisticated reader equipment and vice versa.

The supported complexity of the device containing the ERI data may range from a very simple read-only device that only contains the vehicle's identifier, to a sophisticated device that includes both authentication and confidentiality measures and maintains a historic list of the vehicle data written by the manufacturer and by vehicle registration authorities.

Following the events of 11 September 2001, and subsequent reviews of anti-terrorism measures, the need for ERI has been identified as a possible anti-terrorism measure. The need for international or pan-European harmonization of such ERI is therefore important. It is also important to ensure that any ERI measures contain protection against misuse by terrorists.

This part of ISO 24534 makes use of the basic automatic vehicle identification (AVI) provisions already defined in ISO 14814 and ISO 14816.

Automatic vehicle and equipment identification — Electronic registration identification (ERI) for vehicles —

Part 4: Secure communications using asymmetrical techniques

1 Scope

This part of ISO 24534 provides requirements for electronic registration identification (ERI) that are based on an identifier assigned to a vehicle (e.g. for recognition by national authorities) suitable to be used for:

— electronic identification of local and foreign vehicles by national authorities;

— vehicle manufacturing, in-life maintenance and end-of-life identification (vehicle life cycle management);

— adaptation of vehicle data (e.g. for international resales);

— safety-related purposes;

— crime reduction;

— commercial services.

It adheres to privacy and data protection regulations.

This part of ISO 24534 specifies the interfaces for a secure exchange of data between an ERT and an ERI reader or ERI writer in or outside the vehicle using asymmetric encryption techniques.

NOTE 1 The onboard device containing the ERI data is called the electronic registration tag (ERT).

This part of ISO 24534 includes:

— the application layer interface between an ERT and an onboard ERI reader or writer;

— the application layer interface between the onboard ERI equipment and external ERI readers and writers;

— security issues related to the communication with the ERT.

NOTE 2 The vehicle identifiers and possible additional vehicle data (as typically contained in vehicle registration certificates) are defined in ISO 24534-3.

NOTE 3 The secure application layer interfaces for the exchange of ERI data with an ERI reader or writer are specified in both this part of ISO 24534 and ISO 24534-5.

2 Normative references

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

ISO/IEC 8824 (all parts), *Information technology — Abstract Syntax Notation One (ASN.1)*

ISO/IEC 8825-2, *Information technology — ASN.1 encoding rules: Specification of Packed Encoding Rules (PER) — Part 2*

ISO/IEC 14443 (all parts), *Identification cards — Contactless integrated circuit cards — Proximity cards*

ISO 15628:2007, *Road transport and traffic telematics — Dedicated short range communication (DSRC) — DSRC application layer*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
access control
prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner

[ISO 7498-2:1989, definition 3.3.1]

3.2
access control list
list of entities, together with their access rights, which are authorized to have access to a resource

[ISO 7498-2:1989, definition 3.3.2]

3.3
active threat
threat of a deliberate unauthorized change to the state of the system

[ISO 7498-2:1989, definition 3.3.4]

EXAMPLE Examples of security-relevant active threats may include modification of messages, replay of messages, and insertion of spurious messages, masquerading as an authorized entity and denial of service.

3.4
additional vehicle data
ERI data in addition to the vehicle identifier

[ISO 24534-3:2008, definition 3.1]

3.5
air interface
conductor-free medium between onboard equipment (OBE) and the reader/interrogator through which the linking of the OBE to the reader/interrogator is achieved by means of electromagnetic signals

[ISO 14814:2006, definition 3.2]

3.6
authority
organization that is allowed by public law to identify a vehicle using ERI

3.7
authorization
granting of rights, which includes the granting of access based on access rights

[ISO 7498-2:1989, definition 3.3.10]

3.8
certification authority
natural or legal person trusted to create public key certificates

NOTE See also top-level certification authority and intermediate certification authority.

3.9
challenge
data item chosen at random and sent by the verifier to the claimant, which is used by the claimant, in conjunction with secret information held by the claimant, to generate a response which is sent to the verifier

[ISO/IEC 9798-1:1997, definition 3.3.5]

NOTE In this part of ISO 24534 the term challenge is also used in case an ERT does not have enabled encryption capabilities and the challenge is merely copied without any secret information applied.

3.10
ciphertext
data produced, through the use of encipherment; the semantic content of the resulting data is not available

[ISO 7498-2:1989, definition 3.3.14]

3.11
claimant
entity which is or represents a principal for the purposes of authentication

NOTE A claimant includes the functions necessary for engaging in authentication exchanges on behalf of a principal.

[ISO/IEC 10181-2:1996, definition 3.10]

3.12
cleartext
intelligible data, the semantic content of which is available

[ISO 7498-2:1989, definition 3.3.15]

3.13
confidentiality
property that information is not made available or disclosed to unauthorized individuals, entities, or processes

[ISO 7498-2:1989, definition 3.3.16]

3.14
credentials
data that is transferred to establish the claimed identity of an entity

[ISO 7498-2:1989, definition 3.3.17]

3.15
cryptography
discipline which embodies principles, means, and methods for the transformation of data in order to hide its information content, prevent its undetected modification and/or prevent its unauthorized use

[ISO 7498-2:1989, definition 3.3.20]

3.16
data integrity
integrity
property that data has not been altered or destroyed in an unauthorized manner

[ISO 7498-2:1989, definition 3.3.21]

3.17
decipherment
decryption
reversal of a corresponding reversible encipherment

[ISO 7498-2:1989, definition 3.3.23]

3.18
digital signature
signature
data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery, e.g. by the recipient

[ISO 7498-2:1989, definition 3.3.26]

NOTE See also cryptography.

3.19
distinguishing identifier
information which unambiguously distinguishes an entity

[ISO/IEC 9798-1:1997, definition 3.3.9]

3.20
electronic registration identification
ERI
action or act of identifying a vehicle with electronic means for purposes as mentioned in the scope of this part of ISO 24534

3.21
electronic registration reader
ERR
device used to read or read/write data from or to an ERT

3.22
electronic registration tag
ERT
onboard ERI device that contains the ERI data including relevant security provisions and one or more interfaces to access that data

NOTE 1 In the case of high security, the ERT is a type of SAM (secure application module).

NOTE 2 The ERT can be a separate device or can be integrated into an onboard device that also provides other capabilities (e.g. DSRC communications).

3.23
encipherment
encryption
cryptographic transformation of data to produce ciphertext

NOTE 1 Encipherment may be irreversible, in which case the corresponding decipherment process cannot feasibly be performed.

NOTE 2 Adapted from ISO 7498-2.

3.24
end-to-end encipherment
encipherment of data within or at the source end system, with the corresponding decipherment occurring only within or at the destination end system

[ISO 7498-2:1989, definition 3.3.29]

3.25
entity authentication
corroboration that an entity is the one claimed

[ISO/IEC 9798-1:1997, definition 3.3.11]

3.26
ERI data
vehicle identifying data which can be obtained from an ERT

NOTE ERI data consists of the vehicle identifier and possible additional vehicle data.

3.27
ERI reader
device used to read ERI data directly or indirectly from an ERT by invoking ERI transactions

NOTE 1 In the case that an ERI reader exchanges the ERI protocol data units directly via a data link with an ERT it is also called an ERR. In case it communicates via one or more nodes, only the last node in this sequence is called an ERR. As a consequence, an external ERI reader can, depending on the onboard configuration for example, act for some, but not all, vehicles as an ERR.

NOTE 2 See also onboard ERI reader and external ERI reader.

3.28
ERI transaction
transaction as defined in Clause 6

3.29
ERI writer
device used to write ERI data directly or indirectly into an ERT by invoking ERI transactions

NOTE 1 In case an ERI writer exchanges the ERI protocol data units directly via a data link with an ERT it is also called an ERR. In case it communicates via one or more nodes, only the last node in this sequence is called an ERR. As a consequence, an external ERI writer may, e.g. depending on the onboard configuration, act for some vehicles as an ERR and for others not.

NOTE 2 See also onboard ERI writer and external ERI writer.

3.30
ERT holder
legal or natural person holding an ERT

NOTE The ERT holder could be, for example, the holder of the registration number or the owner, operator or keeper of the vehicle.

3.31
ERT number
number assigned to and written into an ERT that acts as an ERT unique identifier

NOTE The ERT number is assumed to be written into the ERT during its manufacture and once written cannot be changed.

3.32
external ERI reader
ERI reader not being part of the onboard ERI equipment

NOTE 1 An external ERI reader is fitted neither within nor on the outside of the vehicle.

NOTE 2 A distinction is made between proximity, short-range (DSRC), and remote external readers. A proximity reader can be a PCD (proximity coupling device) as specified in ISO/IEC 14443. A short-range external ERI reader may be a part of roadside equipment, handheld equipment, or mobile equipment. A remote external ERI reader may be part of the back office equipment (BOE).

3.33
external ERI writer
ERI writer not being part of the onboard ERI equipment

NOTE 1 An external ERI writer is not fitted within or on the outside of the vehicle.

NOTE 2 A distinction is made between proximity, short-range (DSRC), and remote external writers. A proximity reader can be, for example, a PCD (proximity coupling device) as specified in ISO/IEC 14443. A short-range external ERI writer can be (a part of) roadside equipment, handheld equipment, or mobile equipment. A remote external ERI writer can be part of the back office equipment (BOE).

3.34
hash-code
string of bits which is the output of a hash-function

3.35
hash-function
function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties:

a) for a given output, it is computationally infeasible to find an input which maps to this output; and

b) for a given output, it is computationally infeasible to find a second input which maps to the same output

[ISO/IEC 10118-1:2000, definition 3.5]

NOTE Computational feasibility depends on the specific security requirements and environment.

3.36
identification
action or act of establishing the identity

NOTE See also vehicle identification.

3.37
intermediate certification authority
certification authority for which public key certificates are issued by the top-level certification authority

NOTE This definition implies that there can be only one “level” of intermediate certification authorities.

3.38
key
sequence of symbols that controls the operations of a cryptographic transformation (e.g. encipherment, decipherment, cryptographic check function, signature generation, or signature verification)

[ISO/IEC 9798-1:1997, definition 3.3.13]

NOTE See ISO/IEC 9798-1 for the meaning of the terms used for the examples of cryptographic transformations.

3.39
lifetime
period of time during which an item of equipment exists and functions

NOTE Adapted from ISO 14815.

3.40
manipulation detection
mechanism which is used to detect whether a data unit has been modified (either accidentally or intentionally)

[ISO 7498-2:1989, definition 3.3.35]

3.41
masquerade
pretence by an entity to be a different entity

[ISO 7498-2:1989, definition 3.3.36]

3.42
non-repudiation
property that none of the entities involved in a communication can deny in all or in part its participation in the communication

NOTE Adapted from ISO 7498-2.

3.43
onboard ERI equipment
equipment fitted within or on the outside of the vehicle and used for ERI purposes

NOTE The onboard ERI equipment comprises an ERI and can also comprise any additional communication devices.

3.44
onboard ERI reader
ERI reader which is part of the onboard ERI equipment

NOTE An onboard ERI reader can be, for example, a proximity coupling device (PCD) as specified in ISO/IEC 14443.

3.45
onboard ERI writer
ERI writer which is part of the onboard ERI equipment

NOTE An onboard ERI writer can be, for example, a proximity coupling device (PCD) as specified in ISO/IEC 14443.

3.46
passive threat
threat of unauthorized disclosure of information without changing the state of the system

[ISO 7498-2:1989, definition 3.3.38]

3.47
password
confidential authentication information, usually composed of a string of characters

[ISO 7498-2:1989, definition 3.3.39]

3.48
periodic motor vehicle test
compulsory periodic (e.g. annual) test of the roadworthiness of a motor vehicle of above a specified age, or a certificate of passing such a test

EXAMPLE The MOT test in the United Kingdom is a periodic motor vehicle test.

3.49
principal
entity whose identity can be authenticated

[ISO/IEC 10181-2:1996, definition 3.15]

3.50
privacy
right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed

[ISO 7498-2:1989, definition 3.3.43]

NOTE Because this term relates to the right of individuals, it cannot be very precise and its use should be avoided except as a motivation for requiring security.

3.51
private decipherment key
private key which defines the private decipherment transformation

[ISO/IEC 9798-1:1997, definition 3.3.16]

3.52
private key
key of an entity's asymmetric key pair which should only be used by that entity

[ISO/IEC 9798-1:1997, definition 3.3.17]

NOTE In the case of an asymmetric signature system the private key defines the signature transformation. In the case of an asymmetric encipherment system the private key defines the decipherment transformation.

3.53
private signature key
private key which defines the private signature transformation

[ISO/IEC 9798-1:1997, definition 3.3.18]

3.54
public encipherment key
public key which defines the public encipherment transformation

[ISO/IEC 9798-1:1997, definition 3.3.19]

3.55
public key
key of an entity's asymmetric key pair which can be made public

[ISO/IEC 9798-1:1997, definition 3.3.20]

NOTE In the case of an asymmetric signature system the public key defines the verification transformation. In the case of an asymmetric encipherment system the public key defines the encipherment transformation. A key that is “publicly” known is not necessarily globally available. The key is only made available to all members of a pre-specified group.

3.56
public key certificate
certificate
public key information of an entity signed by the certification authority and therefore rendered unforgeable

[ISO/IEC 9798-1:1997, definition 3.3.21]

NOTE In this International Standard, a public key certificate also specifies the role of the entity for which the public key information is provided, e.g. manufacturer or registration authority.

3.57
public verification key
public key which defines the public verification transformation

[ISO/IEC 9798-1:1997, definition 3.3.23]

3.58
random number
time variant parameter whose value is unpredictable

[ISO/IEC 9798-1:1997, definition 3.3.24]

3.59
registration authority
⟨for vehicles⟩ authority responsible for the registration and maintenance of vehicle records

NOTE The authority may provide vehicle records to accredited organizations.

3.60
registration authority
⟨for ERI data⟩ organization responsible for writing ERI data and security data according to local legislation

NOTE The registration authority for ERI data can be the same as the registration authority for vehicles. This part of ISO 24534, however, does not require this.

3.61
registration certificate
vehicle registration document (paper or smart card) issued by the registration authority for vehicles in which the vehicle and its owner or lessee are registered

3.62
replay attack
masquerade which involves use of previous transmitted messages

[ISO/IEC 9798-1:1997, definition 3.3.26]

3.63
security
protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them

[ISO/IEC 12207, definition 3.25]

3.64
sequence number
time variant parameter whose value is taken from a specified sequence which is non-repeating within a certain time period

[ISO/IEC 9798-1:1997, definition 3.3.27]

3.65
threat
potential violation of security

[ISO 7498-2:1989, definition 3.3.55]

3.66
top-level certification authority
certification authority whose certificates can be verified because its public verification key(s) are written as read-only data into the ERT before the ERT is customized or commissioned

3.67
unilateral authentication
entity authentication which provides one entity with the assurance of the other's identity but not vice versa

[ISO/IEC 9798-1:1997, definition 3.3.33]

3.68
vehicle identification
action or act of establishing the identity of a vehicle

3.69
verifier
entity which is or represents the entity requiring an authenticated identity

NOTE 1 A verifier includes the functions necessary for engaging in authentication exchanges.

NOTE 2 Adapted from ISO/IEC 10181-2.

4 Abbreviations

AEI	automatic equipment identification
AES	advanced encryption standard
ASN.1	Abstract Syntax Notation One [as defined in ISO/IEC 8824 (all parts)]
AVI	automatic vehicle identification
BOE	back office equipment
EN	Europäische Norm (German), English: European Standard
ENV	Europäische Norm Vorausgabe (German), English: European Pre-Standard
ERI	electronic registration identification
ERR	electronic registration reader
ERT	electronic registration tag
EU	European Union
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
OBE	onboard equipment (including non-ERI equipment)
OSI	Open Systems Interconnection (see ISO/IEC 7498-1)
PICS	protocol implementation conformance statement(s)
PIN	personal identification number
SAM	secure application module
VIN	vehicle identification number

5 System communications concept

5.1 Introduction

This clause is informative only. It provides an introduction to the context in which ERI data and security data may be read from, or written into, the ERT and in which vehicles can be identified. It also outlines options that may or may not be used in an actual implementation. The normative requirements for the application layer interfaces are provided in Clause 6 and Annex A. Annex B contains a form to specify the limitations of an actual communication protocol implementation.

5.2 Overview

5.2.1 Vehicle registration identification

ERI (electronic registration identification) is the action or act of identifying a vehicle by electronic means for the purposes mentioned in the scope of this part of ISO 24534.

The identifier used to identify a vehicle is called the vehicle identifier or vehicleId.

NOTE 1 The preferred vehicle identifier is the VIN that is assigned to the vehicle by its manufacturer in accordance with ISO 3779, but alternatives are supported as well (see ISO 24534-3 for details).

NOTE 2 See ISO 24534-3 for details about the vehicle identifier and ERI data.

In this part of ISO 24534, the combination of the almost unique vehicleId and a unique ERT number is used as the unambiguous distinguishing identifier.

5.2.2 System concept and supported interfaces

Figure 1 presents the interfaces for which the application layer is specified in this part of ISO 24534.

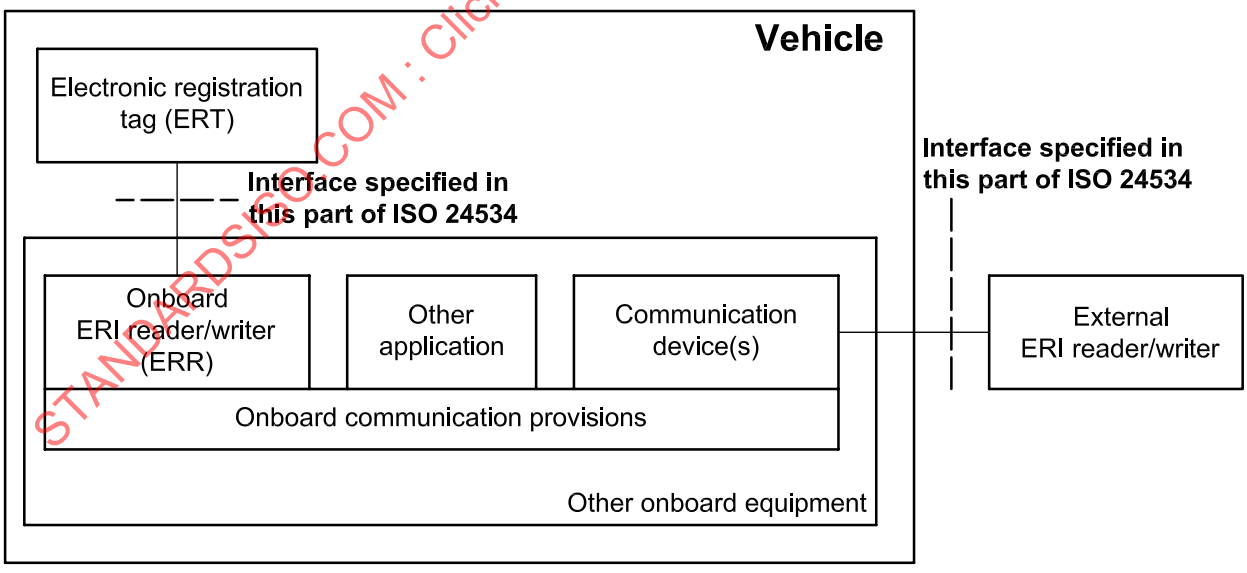

Figure 1 — System concept and supported interfaces

The onboard component that provides a secure environment for the ERI data and security data is called the ERT (electronic registration tag).

NOTE 1 An implementer may integrate other provisions (e.g. additional communication provisions) into an ERT as long as this does not compromise the security of the ERT.

Depending on its capabilities, an ERT is tailored to a specific vehicle in three consecutive steps (see Figure C.2).

a) First, it is customized with the vehicle identifier and, optionally, additional vehicle data. This step can only be performed once in the lifetime of an ERT. Customizing does not yet enable any ERT encipherment or signing services.

b) Second, a registration authority may commission itself as the registration authority for the vehicle by adding its security data. This step may be performed at any time by a registration authority when it wants to commission itself as the current registration authority. A registration authority may change its security data by recommissioning itself. If supported by the ERT, commissioning enables ERT confidentiality and authentication services by providing the required security keys. If a key is not provided, the corresponding services will not be enabled.

NOTE 2 Most smart cards are “owned” and “controlled” by one issuer during their whole lifetime. For an ERT this is more complicated. When a vehicle is sold to another country a new registration authority will take over the “control” or “ownership” of the ERT when it issues a new number plate and a new registration certificate for the vehicle. The ERT is then recommissioned.

c) Third, a registration authority that has commissioned itself may change the additional vehicle data to register a change of the vehicle data (with the exception of the vehicleId).

NOTE 3 In order to accommodate the needs of different countries, different selections of additional vehicle data can be included. (See ISO 24534-3 for details.)

The onboard communication provisions shall be capable of transferring data from or to the ERT without modifying that data.

NOTE 4 The onboard communication provisions may e.g. be part of an onboard platform for transport applications.

A communication device may communicate with an external proximity reader or writer, with a short-range ERI reader and/or writer, or with remote back office equipment (BOE).

A communication device that communicates with an external ERI reader/writer acts as a relay between this external ERI reader/writer and the onboard ERI reader/writer. A communication device may also be used for other applications.

5.2.3 Roles involved

Within the context of this part of ISO 24534 the following “roles” for natural or legal persons are distinguished.

— Manufacturers, who assign a VIN or chassis number to each vehicle they build. A manufacturer may also once customize an ERT for a particular vehicle.

— Registration authorities (with respect to the ERI data), which may:

  — assign a new vehicleId to the vehicle (in the case of defects) and may customize an ERT (e.g. in the case of defects or retrofitting),

    NOTE 1 A registration authority can assign a new vehicle ID to the vehicle (e.g. when the number on the chassis has become corrupted). Then it will put that new ID on the chassis and write it in a new ERT (a vehicle ID can never be overwritten).

  — commission themselves as the registration authority for a vehicle,

  — authorize other authorities to read the ERI data,

  — authorize an ERT holder to grant additional service providers access to ERI data, and which

— which are responsible for the registration of additional vehicle data into an ERT according to local legislation (see below for details).

NOTE 2 It is expected that the registration authority with respect to the ERI data is the same authority that keeps the official register in which the vehicle is listed. This is however not required by this part of ISO 24534.

NOTE 3 It is assumed that each vehicle is listed in a register that contains the vehicle identifier and additional data related to the vehicle. It is implicitly assumed that this register also identifies the one(s) responsible for the vehicle (e.g. its owner, operator, keeper, lessee, and/or regular driver).

— Certification authorities, which are trusted to create public key certificates (referred to as “certificates” in this part of ISO 24534). Public key certificates are used to prevent a fraudulent organization from disguising itself as a manufacturer or registration authority. There are two types of certification authorities:

— one top-level certification authority, and

— zero or more intermediate certification authorities.

NOTE 4 A certification authority will not directly communicate with the ERT. Their certificates are used by manufacturers and registration authorities.

NOTE 5 With two levels of certification authority, the top-level authority can delegate the distribution of certificates to an intermediate authority, which is then responsible for creating the certificates for registration authorities and manufacturers within some region (e.g. the states of the United States or the member states of the EU).

— Authorities, which are authorized by the registration authority to read the ERI data from a vehicle (e.g. because they are entitled to do so by virtue of public legislation).

— Additional service providers (public or private), which provide a service that requires an electronic identification of a vehicle and/or certificated vehicle data. The ERT holder may or may not authorize an additional service provider to read the vehicle's identifier and the additional vehicle data.

— ERT holders, which are holding the ERT. An ERT holder may e.g. be the holder of the vehicle's registration number or the owner, operator, or keeper of the vehicle.

Even in cases where confidentiality of the ERI data is supported, the ERT holder is entitled both to read the ERI in its vehicle and to allow other service providers to read the vehicle identifier. A PIN issued by the registration authority to the ERT holder provides the required access control for the ERT holder.

NOTE 6 Roles and requirements related to the specification, design and manufacturing (including testing) of an ERT are outside the scope of this part of ISO 24534.

5.2.4 The communications context for reading

Figure 2 shows the communications context for reading data from an ERT.

An onboard or an external ERI reader is used to read data from the ERT. An onboard ERI reader communicates directly with the ERT. An external ERI reader may communicate either directly or indirectly with the ERT, e.g. directly in the case of a handheld reader or an integrated ERI device, or indirectly via an onboard communication module and the onboard ERI reader. The onboard communication module may also be used for other applications.

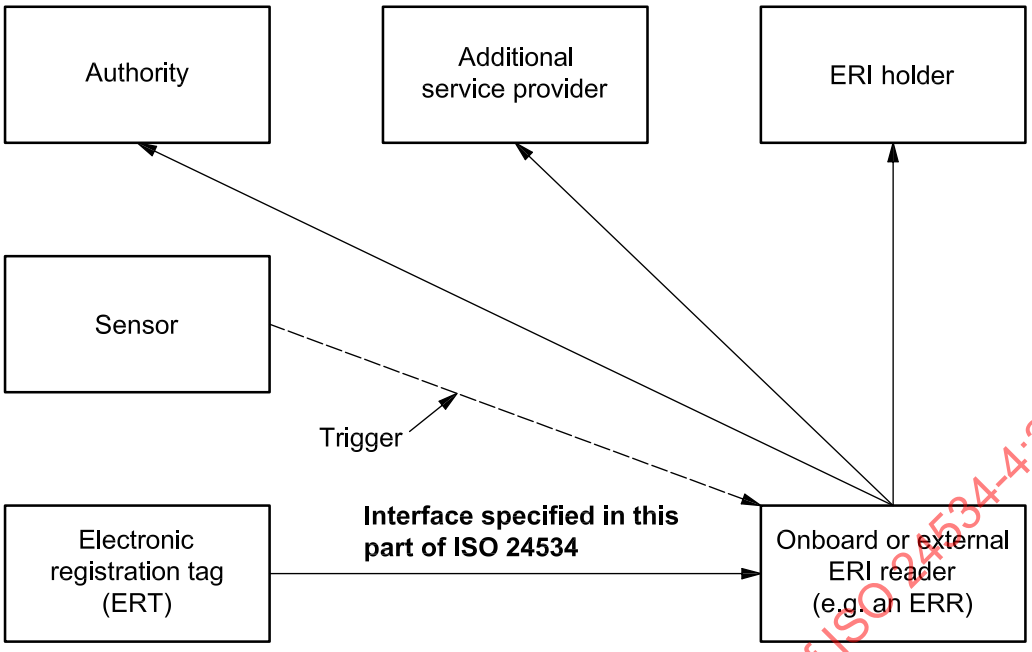

Figure 2 — Communication context for reading from an ERT

A sensor system (outside the scope of this International Standard) may be used to trigger an external ERI reader when it senses the presence of a vehicle that needs to be identified.

The various parties that can read ERI data from an ERT are described in 5.2.3. The access rights of the various entities are described in 5.3.5.1.

An ERT holder may wish to have access to the ERI data for various reasons:

— to verify the correctness of the ERI data;

— to obtain an authenticated (i.e. signed) vehicle identifier or ERI data to be used for another application;

— to verify the access control list (see 5.3.5), if present in the ERT;

— to verify the historic ERI data and/or historic commissioning data, if present in the ERT.

The equipment used by an authority, additional service provider or ERT holder in its office (i.e. not at the roadside) is called back office equipment (BOE).

The distribution of functions between BOE and an external ERI reader is outside the scope of this International Standard.

5.2.5 The communications context for writing

Figure 3 presents the communications context for writing data into an ERT.

The onboard or external ERI writer is used to write data into the ERT. An onboard ERI writer communicates directly with the ERT. An external ERI writer may communicate either directly or indirectly with the ERT, e.g. directly in the case of a handheld writer or an integrated ERI device, or indirectly via an onboard communication module and the onboard ERI writer. The onboard communication module may also be used for other applications.

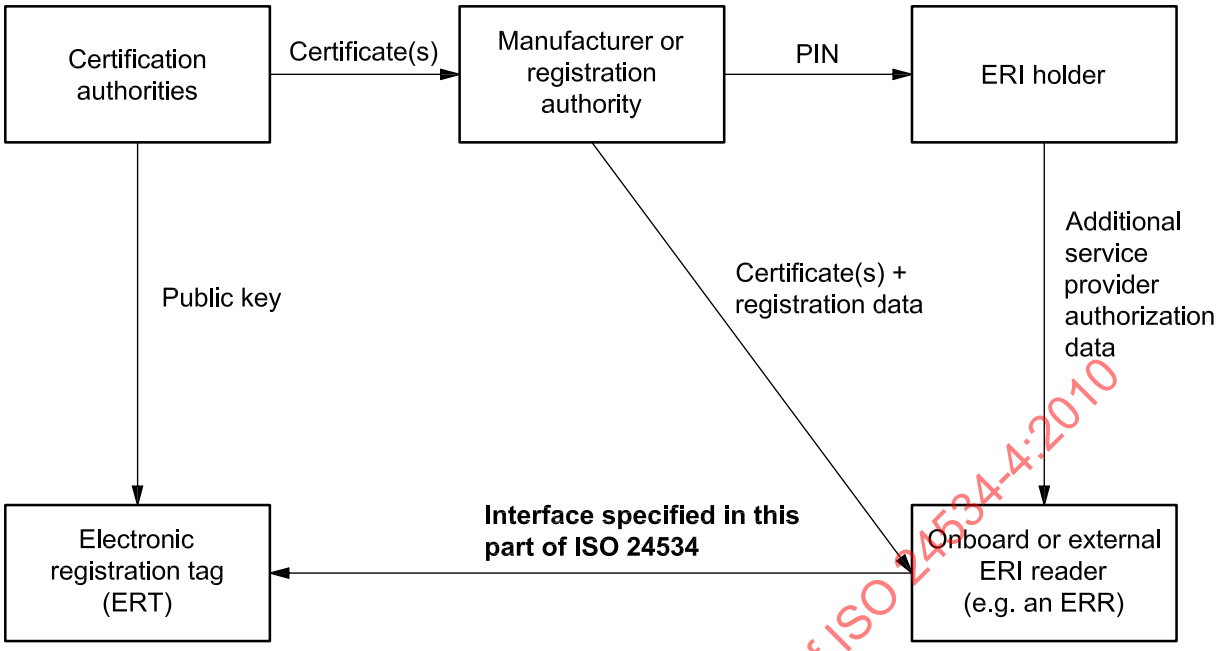

Figure 3 — Communication context for writing into an ERT

The various parties that can write ERI (security) data into an ERT are described in 5.2.3. The access rights of the various entities are described in 5.3.5.2.

The certification authorities provide public key certificates for genuine manufacturers and genuine registration authorities. The certificates are used while writing data into the ERT in order to prove that (signed) data originates from a genuine manufacturer or genuine registration authority. The certificates are also used while the ERI data is read in order to check that the data received stems from a genuine ERT, manufacturer or registration authority.

The registration authority authorizes the ERT holder by providing a password (PIN). The ERT holder may then authorize an additional service provider to read ERI data.

The distribution of functions between BOE and an external ERI writer is outside the scope of this International Standard. A registration authority may e.g. commission a writer to operate on its behalf or it may use e.g. the writer only as a relay device for remote writing from its back office.

5.2.6 ERT capability levels supported

It is envisioned that different users may have different requirements for ERI. Therefore, this part of ISO 24534 supports different service levels of ERT capabilities.

Figure 4 presents an overview of the various capability levels supported by this part of ISO 24534. See also the protocol implementation conformance statements (PICS) pro forma in Annex B.

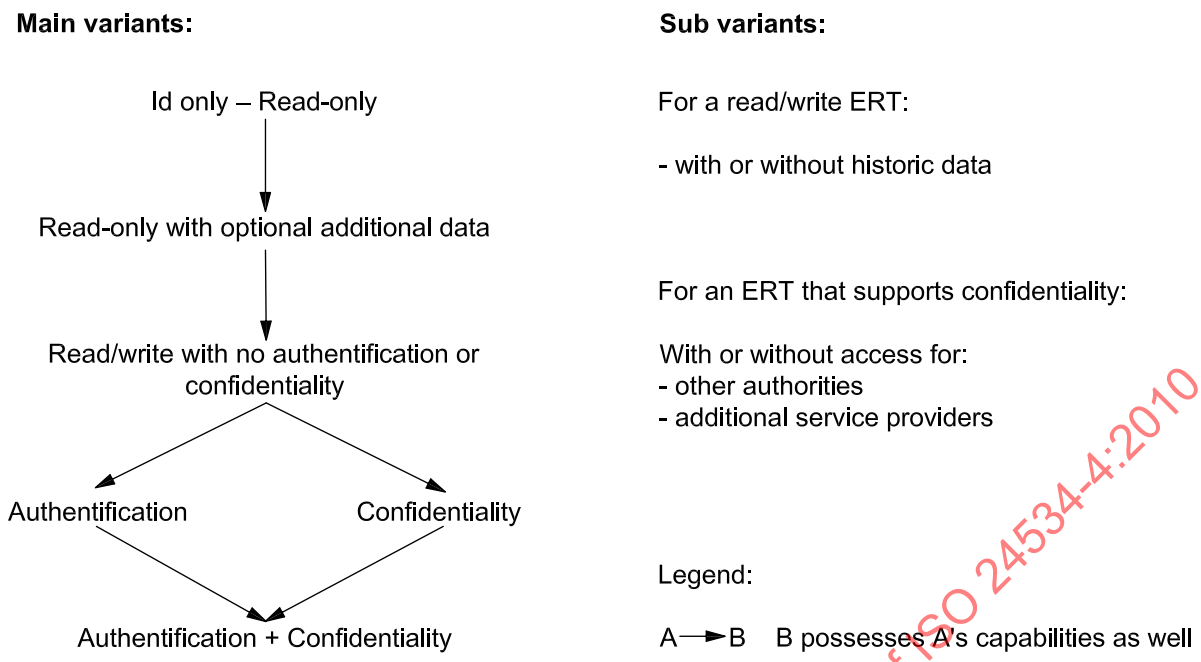

Figure 4 — ERT capability levels supported

The following main variants are distinguished (a variant at the arrowhead also includes the capabilities of the one at the arrow tail):

— Id only – Read only: this is the simplest ERT. The vehicle identifier is written only once and no other information can be added. In this case an ERI reader can only read the vehicle identifier from the vehicle (ERT).

— Read only with optional additional vehicle data: in this case the ERT may also contain additional vehicle data as specified in ISO 24534-3. However, all data can only be written once in the lifetime of an ERT. For security reasons the data may also contain a signature from the origination entity.

— Read/write with no authentication or confidentiality: in this case the ERT contains the vehicle identifier plus optional additional vehicle data. This additional vehicle data can be updated if required.

— Read/write with authentication: in this case the ERT also provides authentication services. The ERT can check a signature attached to data written into it. If enabled, the ERT can also attach a signature to data it sends to an ERI reader. An ERI reader can then check that the data it receives stems from a genuine ERT and not from a fake one.

NOTE 1 A registration authority can disable the ERT signing capabilities but not the signature verification capabilities.

— Read/write with confidentiality: in this case the ERT is capable of providing confidentiality services. The ERT can decrypt security data encrypted by an ERD writer. If enabled, the ERT can also encrypt the data it sends to an ERI reader. This data can only be interpreted by those parties which possess the corresponding private decipherment key.

NOTE 2 A registration authority can disable the ERT encipherment capabilities but not the decipherment capabilities.

— Read/write with both authentication and confidentiality: in this case the ERT provides both authentication and confidentiality services.

In addition to these main variants, various subvariants may be distinguished, e.g.:

— A read/write ERT which also maintains a log of historic data, i.e. of previous write operations;

— An ERT with confidentiality services for which the registration authority can also authorize other authorities to read the ERI data;

— An ERT with confidentiality services for which the registration authority may also grant the ERT holder permission to read the (historic) ERI data in their ERT;

— An ERT with confidentiality services for which the registration authority may also grant the ERT holder permission to authorize additional service providers to read the ERI data in their ERT.

5.2.7 Security service levels and interoperability

This part of ISO 24534 supports authorities in identifying vehicles roaming from the jurisdiction of other registration authorities. Therefore, much attention is given to the interoperability of ERTs and readers of various capability levels, i.e. the identification of a vehicle with a simple ERT by a sophisticated reader and vice versa.

The basic principle for the interoperability of ERTs and ERI readers is that the reader will function with the full range of capabilities of the ERT. If the reader does not provide full functionality, e.g. the required decipherment capabilities, then the authority using the reader may ask for (decipherment) assistance from the registration authority. A similar situation applies when the ERT signs data.

Table 1 presents the various combinations when an ERT and an ERI reader have different cryptographic capabilities.

Table 1 — Interoperability between ERTs and ERI readers

ERT	ERI reader	Interoperability
Encipherment	Decipherment	
Yes	No	The reader will not understand the ERI data, which needs the assistance of the vehicle's registration authority for decrypting the data read.
No	Yes	The reader does not need to use its decryption capabilities.
Signing	Signature verification	
Yes	No	The reader will not be able to check any signature provided for the ERI data.
No	Yes	The reader does not need to use its signature verification capabilities.

In order to support readers with no decipherment capabilities, a registration authority may, if required, disable the encipherment capabilities of ERT for vehicles in its jurisdiction.

EXAMPLE When a vehicle is imported from a country with strict privacy protection into a country with less strict privacy protection and which consequently employs readers with no decipherment capabilities, then the registration authorities may disable the decipherment capabilities of the ERT when it is commissioned.

NOTE 1 Harmonization of the security requirements of the registration authority is outside the scope of this International Standard.

With respect to the interoperability between ERTs and ERI writers, precedence has been given to security above the support of international (re)sales. As a registration authority may replace an ERT, this is not considered to be a severe restriction. The basic principle for writing is that a writer and, hence, a registration authority can determine the capabilities of an ERT and, based upon this, the registration authority can decide to accept that ERT or replace it with one that matches its requirements.

NOTE 2 If a registration authority does not accept a particular ERT, then that authority should replace it with one that is acceptable and customize it as necessary.

The interoperability of ERTs and ERI writers of various capability levels is depicted in Table 2.

Table 2 — Interoperability between ERTs and ERI writers

ERI writer	ERT	Interoperability
Encipherment	Decipherment	
Yes	No	The ERI writer can only write data into such an ERT if it disables its encipherment capabilities.
No	Yes	The ERT will not accept any data that has to be encrypted from such a writer. More precise, the ERT will then not accept any private signature key or PIN in a commissioning transaction. As a consequence the invocation of a commissioning transaction will disable the ERT signing capabilities and will prevent the use of a PIN by an ERT holder.
Signing	Signature verification	
Yes	No	The ERT will not verify any signature provided by the writer. As a consequence every writer can be used to change the ERI data and its security data. NOTE This is not advisable, but allowed in this International Standard. It is the responsibility of the registration authority to allow or not to allow (i.e. to commission or not to commission) this kind of ERT.
No	Yes	The ERT will not accept any data from this writer.

It is the responsibility of a registration authority to accept or not to accept a particular ERT within its jurisdiction, i.e. a registration authority may or may not want to commission itself on a particular ERT.

5.3 Security services

5.3.1 Assumptions

The security concept for the exchange of data between an ERT and an ERI reader or writer is based on the following assumptions.

— The use of an ERT may be mandatory and, therefore, should be fraud resistant. Using ISO 7498-2 terminology, the ERT should be resistant against active threats (e.g. modification of messages, replay of messages, insertion of spurious messages, and masquerading).

— A reading of an ERT should be suitable as legal evidence (non-repudiation).

NOTE This may require the specification of a protection profile (including an adequate evaluation assurance level, EAL) for the ERT according to ISO 15408. However, such a specification is outside the scope of this International Standard.

— ERI shall have the capability to provide a high level of privacy protection (i.e. it should not be easily possible to monitor mobility patterns of a vehicle and, hence, of its regular driver). As a consequence, an ERT should also be resistant against passive threats.

— ERI shall have the capability to provide protection measures to prevent ERI from being used to trigger an attack on a vehicle.

— The performance of security mechanisms shall be achievable within the time available for communications whilst the vehicle is moving.

EXAMPLE Reading a vehicle at 180 km/h within a 10 metre read range should be achieved within 200 ms.

5.3.2 Entity authentication while reading ERI data

Trust in the authenticity of an ERI reading depends on the following authentication aspects, which must all be fulfilled to fully trust a reading:

a) The ERT is customized with the correct vehicle identifier and is attached to the correct vehicle.

b) The ERT cannot be removed from the vehicle without rendering it inoperable.

c) The ERI data is read from a genuine ERT, i.e. from a legitimate device (it is not a replicated message from a fake one). This is achieved by signing the ERI data together with a challenge code provided by the ERI reader.

d) The ERI data is correctly read from the ERT (data integrity, manipulation detection). This is achieved by standard mechanisms used in data communications, by signing data, and, as a sideeffect, by encrypting the ERI data (decipherment of corrupted ciphertext will not produce anything useful).

e) When ERI data has been correctly read from a genuine ERT upon a particular request it shall be difficult to be disputed later on that this data was not read from this component upon that request (non-repudiation). This is achieved by signing the ERI data together with a challenge code provided by the ERI reader.

NOTE 1 This part of ISO 24534-4 only deals with c), d), and e). Item b) is specified in ISO 24534-2. Item a) is not dealt with in this part of ISO 24534-4.

NOTE 2 Using ISO/IEC 9798-3 terminology, c) and e) are supported using a two-pass unilateral authentication mechanism. Uniqueness/timeliness is then controlled by generating and checking a random number sent by the ERI reader (the verifier) to the ERT (the claimant).

5.3.3 Confidentiality while reading ERI data

This International Standard supports confidentiality by delivering ERI data in an encrypted format. The encrypted ERI data can then be made freely available but can only be decrypted and interpreted by authorized persons/equipment (end-to-end encipherment).

To prevent encrypted ERI data from being able to be used as a pseudonym, a sequence number is added to the ERI data before encryption.

NOTE Confidentiality is, in more technical terms, provided by means of asymmetric keys. In the case of read operations the ERT uses a public encipherment key of an authority written into the ERT by the registration authority or by the ERT holder (for an additional service provider). In the case of write operations, confidentiality is achieved with a public encipherment key which can be obtained from the ERT. The data to be written can then only be decrypted with the private decipherment key of the ERT.

5.3.4 Keys for authentication and confidentiality

A vehicle may be registered for many years and during those years many other vehicles are registered or deregistered. As a consequence a registration authority has either to always use the same keys, or to use different keys for different vehicles. In order to support this latter option a key can be identified with a key identifier.

In the case of encryption, the identifier of the public encipherment key is attached to the ciphertext and can then be used by a recipient to determine the corresponding private decipherment key.

For signature verification one or two public key certificates shall be appended to the signed data. For data signed by the ERT the certificate is provided by the registration authority when the private signature key is written into the ERT (see the commission ERT transaction). For data to be signed by the back office equipment the private signature key for the top level certificate is determined by means of a key identifier that can be obtained from the ERT. This key identifier and the public verification key are written into the ERT when it is manufactured.

This part of ISO 24534 allows a registration authority to change its public encipherment key, the private signature key and the PIN for the ERT holder. This may e.g. be part of a periodic motor vehicle test.

NOTE There is no implication that the garage performing the periodic test will change this key itself. Although, it is outside the scope of this part of ISO 24534 to prescribe the security measures to be applied by the registration authorities to protect their private keys, it is more likely that the garage will only establish a connection between the ERT and the registration authority; the registration authority may then securely update the ERT.

5.3.5 Access control to ERI data

5.3.5.1 Read access control

5.3.5.1.1 Entities with read access

The following three types of entities may require access to ERI data:

— authorities,

— authorized service providers, and

— the ERT holder.

In case the ERT has enabled encipherment capabilities, it controls read access of authorities and authorized additional service providers by means of an “access control list” that contains for each entity the following data:

— its identifier,

— its public encipherment key and key identifier, and

— an indication of whether or not this entity is an authority.

The public encipherment key is used for the encryption of a random secret key used to encrypt the ERI data sent to the ERI reader.

In case a registration authority does not support confidentiality, its public encipherment key will have a null value and any contents of the access control list will be ignored. In that case the ERI data will be sent in cleartext to the ERI reader.

5.3.5.1.2 Read access for authorized authorities

In case an ERT has enabled encipherment capabilities, a registration authority may authorize an authority for reading ERI data by writing its identifier, public encipherment key and key identifier into the access control list of the ERT (see 5.3.5.2).

The ERI reader of an authorized authority can have confidential access to the ERI data by providing the identifier of this authority. The ERT will then encrypt the ERI data with its public encipherment key.

5.3.5.1.3 Read access for unauthorized authorities

An authority that is not authorized to read the ERI data shall not provide any entity identifier in its read request. The ERT will then encrypt the ERI data with the public encipherment key of the vehicle's registration authority.

Decrypting this ERI data then requires the private decipherment key of the vehicle's registration authority. This can be accomplished in several ways (which are all outside the scope of this International Standard):

— The ERI reader can be commissioned providing the private decipherment key of registration authority in e.g. a SAM, in which case the reader can decrypt the ERI data itself.

— The local (registration) authority may have (a SAM with) this private key and may then decrypt the ERI data obtained by the reader.

— The vehicle's registration authority may be requested to decrypt the ERI data.

NOTE Procedures for the cooperation between registration authorities are outside the scope of this International Standard.

5.3.5.1.4 Read access for authorized service providers

Access of a service provider is controlled by adding its identifier, public encipherment key and key identifier to an access control list in the ERT. A reader of this service provider can then be commissioned with the corresponding private decipherment key (in a SAM).

When requesting the ERI data from a vehicle, the reader provides the ERT with the identifier of the service provider. If the access control list contains this identifier, the associated public encipherment key is used for the encryption of the ERI data returned to the reader. If not, the public encipherment key of the vehicle's registration authority is used.

5.3.5.1.5 Read access for the ERT holder

This International Standard provides access to cleartext ERI data if the request contains both the VehicleId and the password (PIN) supplied to the ERT holder.

Using only the VehicleId violates the requirement that the ERI shall not be easily usable for triggering an attack on the vehicle. The success of an attempt to read data using only the VehicleId as authentication information may then be used as such a trigger. The use of a PIN reduces this risk.

5.3.5.2 Write access control

5.3.5.2.1 Entities with write access

Write access may be required by the following types of entities:

— manufacturers and registration authorities in order to customize the ERT for a particular vehicle,

— registration authorities in order to commission and to update ERI data or security data, and

— the ERT holder.

5.3.5.2.2 Write access for customizing by manufacturers and registration authorities

This International Standard supports customizing an ERT for a particular vehicle by a manufacturer or a registration authority.

Customizing the ERT for a particular vehicle is accomplished by writing the vehicle identifier and, eventually, additional vehicle data (e.g. the registration date, registration number, engine type, etc.) into this component.

An ERT can only be customized once in its lifetime and the vehicle identifier in the ERT can then never be changed.

NOTE 1 This may be accomplished, for example, by a manufacturer during the vehicle build or by a registration authority if the ERT has to be replaced (e.g. because of a defect) or is to be retrofitted.

To obtain this write access the writing equipment shall prove that it is acting on behalf of a manufacturer or a registration authority. To this end the writing equipment shall supply one of two public key certificates:

— a public key certificate issued by either the top-level or an intermediate certification authority, or

— in case the public key certificate is issued by an intermediate certification authority, a public key certificate for this intermediate certification authority issued by the top-level authority.

NOTE 2 As the public verification key(s) of the top-level certification authority are put into the ERT at manufacturing, this component can verify the signature on a certificate issued by the top-level certification authority.

NOTE 3 As it is only required that the data to be written into the ERT be issued by an official registration authority (and contain the correct vehicle identifier) there is no need to authorize the writing equipment. The communication channel used to exchange the ERI data from the BOE of the registration authority to the ERT may be insecure.

5.3.5.2.3 Additional write access for registration authorities

In addition to customization, this International Standard also supports, as far as applicable, write access of a registration authority to the ERT, in order to:

— commission itself as the vehicle's registration authority;

— add or change the private signature key to be used by the ERT to sign messages and the certificate for the corresponding public verification key signed by itself;

NOTE 1 This certificate also includes the vehicle identifier.

— add or change its public encipherment key required for the encipherment of ERI data (if confidentiality is to be supported);

— authorize other authorities by adding their identifier, public encipherment key and key identifier into the ERT's access control list (if confidentiality is to be supported);

— add or change the PIN issued to the ERT holder (if confidentiality is to be supported);

NOTE 2 How and when a registration authority sends an ERT holder a (new) PIN is outside the scope of this International Standard.

— change the additional vehicle data (e.g. its registration number, its colour, its engine type, etc.) as required or allowed by local legislation.

Write access is obtained by handing over one or two public key certificates as described in the previous section.

5.3.5.2.4 Write access for ERT holders

This part of ISO 24534 supports write access of an ERT holder to authorize read access for

— an additional service provider by writing the identifier, public encipherment key and key identifier of this service provider into the ERT, and

— onboard applications/equipment that needs to verify or authenticate ERI data by writing the identifier, public encipherment key and key identifier into the ERT.

To obtain write access the ERT holder shall identify themself as the holder of the ERT containing the ERI data. They do so by using the vehicle identifier and a password (PIN), obtained from the registration authority.

NOTE How and when an ERT holder obtains this password (PIN) is outside the scope of this International Standard.

Using only the vehicle identifier violates the requirement that the ERI shall not be easily usable for triggering an attack on the vehicle. The acceptance of an attempt to write data using only the vehicle identifier as authentication information may then be used as such a trigger. The use of a PIN reduces this risk.

5.3.6 Public certification key identification

The top-level certification authority may use different private signature keys. A key identifier is used to determine the private signature key that corresponds to the public verification key stored in the ERT. This key identifier is stored in and can be read from the ERT.

5.4 Communication architecture description

5.4.1 Identifying a vehicle with an authorized short-range reader

Figure 5 presents the communication concept for the identification of a vehicle with an authorized ERI reader using short-range communications.

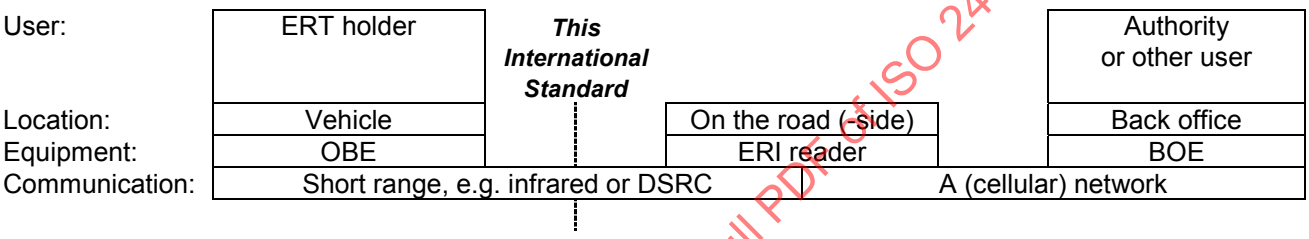

Figure 5 — Short-range communication with an authorized reader

This International Standard deals with the application layer of the air interface between the onboard ERI equipment and external short-range ERI readers.

NOTE The vehicle-external ERI-reader interface corresponds to the DELTA reference point, the air interface, in Annex A of ISO 14814:2006 (see 5.5.1 for details). The external ERI-reader-back-office interface corresponds to the ALPHA reference point in that annex.

The interface between an external ERI reader and the BOE of a back office is outside the scope of this International Standard. It may e.g. be used for commissioning the ERI reader, the exchange of white or black lists and/or uploading the reading results. It may e.g. be a local interface in the back office or a wide area network interface.

A back office may be an office from a (registration) authority or from an additional service provider.

A similar situation applies to, and a similar figure can be drawn for, the identification of a vehicle with an authorized external proximity reader.

5.4.2 Identifying a vehicle with a non-authorized short-range reader

Figure 6 presents the communication concept for the identification of a vehicle with a non-authorized short-range ERI reader, i.e. with an external ERI reader that does not have a private decipherment key for the decipherment of the ERI data.

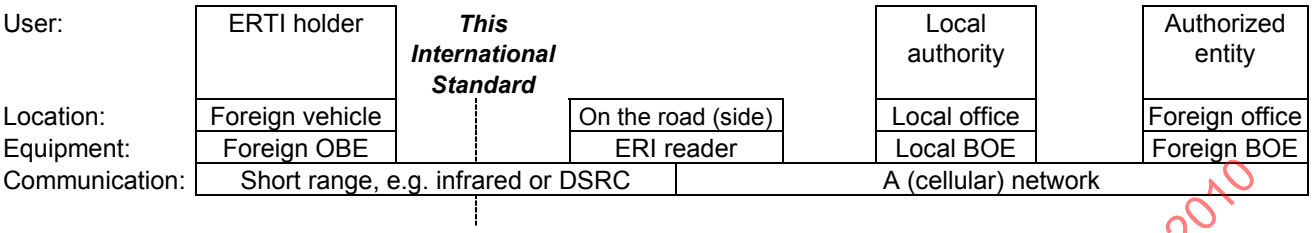

Figure 6 — Short-range communications with a non-authorized reader

Identification of a foreign vehicle, i.e. a vehicle for which the external ERI reader is not authorized, requires the cooperation of an authorized entity, e.g. the registration authority at which the vehicle is registered. This cooperation, however, is outside the scope of this International Standard. It may be used for decrypting the data read from the vehicle and/or obtaining additional information.

The reader may e.g. directly communicate with the foreign BOE or may archive it via its own local BOE.

A similar situation applies to, and a similar figure can be drawn for, the identification of a vehicle with a non-authorized external proximity reader.

5.4.3 Overall communication concept for remote access

This International Standard also supports remote access to an ERT. A registration authority may e.g. use remote access, if implemented, to check or update the additional vehicle data or the security data. An ERT holder may use remote access to check the ERI data or to authorize an additional service provider.

Figure 7 presents the communication concept for remote access to a vehicle's ERT.

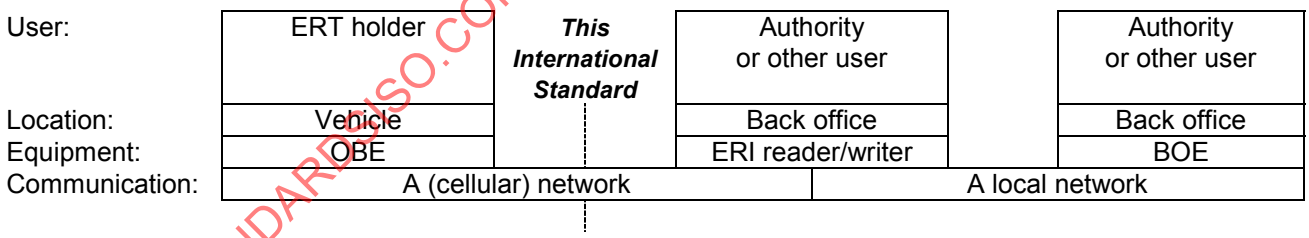

Figure 7 — Overall remote communication concept

This International Standard deals with the application layer of the network interface between the onboard ERI equipment and a remote external ERI reader/writer.

NOTE Whether or not remote access capabilities are implemented is outside the scope of this International Standard.

5.4.4 The onboard communication

Figure 8 presents an abstract overview of a possible onboard communication architecture.

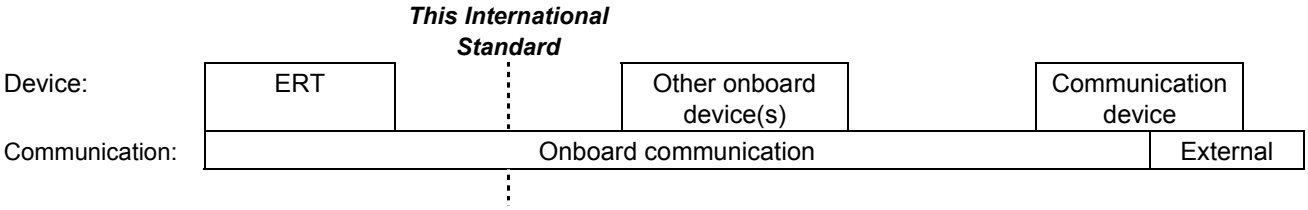

Figure 8 — The onboard architecture

NOTE The figure above does not imply that the ERT and the communication device shall be separate components. This may or may not be the case for a specific implementation.

5.5 Interfaces

5.5.1 The short-range air interface

The communication between the onboard ERI equipment and a short-range external ERI reader/writer uses the following protocol stack:

AVI layer (conforming to ISO 14816 plus additional services)
ERI layer (adding ERI security and management services)
An application layer, e.g. the DSRC application layer (conforming to ISO 15628) or a similar layer
Lower layers

Figure 9 — Protocol stack for the short-range air interface

NOTE The relation between these layers and the reference points BETA to ZETA in informative Annex A of ISO 14814 is depicted in the adapted Figure 10. (Reference point ALPHA is located between the ERI reader and the BOE of a back office.)

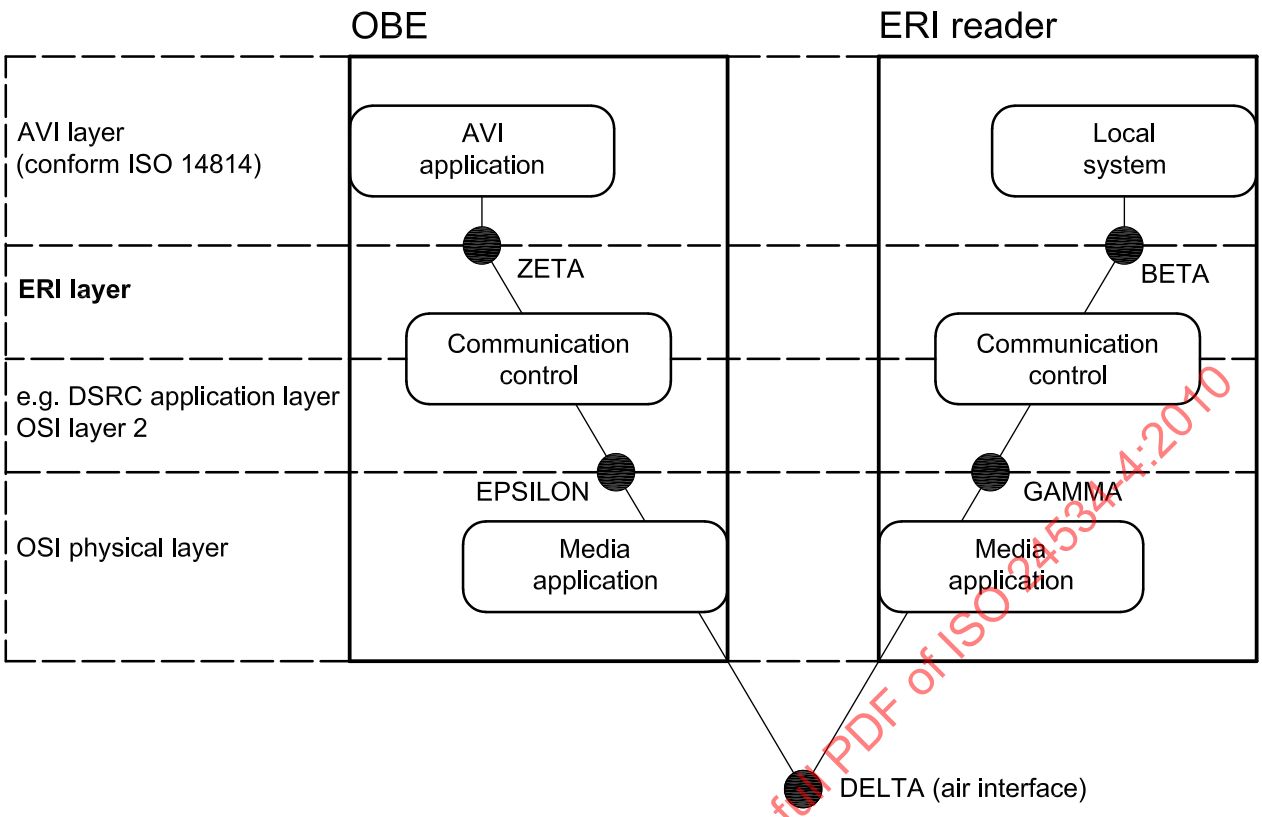

Figure 10 — The location of the ERI layer in the ISO 14814 reference architecture

5.5.2 The proximity interface with the ERT

The communication between an ERT and an onboard or external proximity reader/writer uses the following protocol stack:

AVI layer (conforming to ISO 14816 plus additional services)
ERI layer (adding ERI security and management services)
A transmission layer, e.g. ISO/IEC 14443
Lower layers, e.g. ISO/IEC 14443

Figure 11 — Protocol stack for the interface between an ERT and a proximity reader

6 Interface requirements

6.1 Overview

This clause defines the application layer interface to access the ERI data in the ERT and in particular:

— Subclause 6.2 provides an abstract definition of the ERI transactions using ASN.1,

— Subclause 6.3.2 defines the proximity interface with the ERT,

— Subclause 6.3.3 defines the short-range air interface between the onboard ERI equipment and an external ERI reader/writer, and

— Subclause 6.3.4 defines the interface for remote access.

The application layer of the onboard interface is defined as an implementation of the abstract definitions in 6.2. The external interfaces as defined in 6.3.3 and 6.3.4 are either used to communicate directly with the ERT in the case where the onboard ERI equipment is integrated into a single device (ERT) or are used to relay ERI protocol data units to the onboard ERI reader/writer (see Figure 1).

6.2 Abstract transaction definitions

6.2.1 Transaction overview

In this clause the ERI transactions in Table 3 are defined.

Table 3 — Required and optional transactions

Clause	Transaction	Req[a]	Description
ERI data reading transactions			
6.2.3	getEriData	R	For ERI by authorities and additional service providers.
6.2.4	getAuthenticatedEriData	C	To obtain authenticated ERI data by the ERT holder.
ERI data management transactions (including customization)			
6.2.5	setEriData	R	For customizing an ERT by a manufacturer or a registration authority and for updating the additional vehicle data by a registration authority.
6.2.6	getCiphertextHistoricEriData	O	To retrieve the argument of earlier setEriData transactions in ciphertext by an authority.
6.2.7	getCleartextHistoricEriData	O	To retrieve the argument of earlier setEriData transactions in cleartext by the ERT holder.
Commissioning transactions			
6.2.8	getPublicCertificateVerificationKeyId	A, C	To retrieve the identifier of the public verification key of the top-level certification authority.
6.2.9	getPublicEnciphermentKeyErt	A, C	To retrieve the public encipherment key of the ERT for the encipherment of the private data component of the commissioning data.
6.2.10	commissionErt	A, C	For a registration authority to commission itself as the registration authority for an ERT and for updating the security data of the registration authority.
6.2.11	withdrawCommissioning	O	For a registration authority to withdraw its commissioning ("end-of-life" support).
6.2.12	getCiphertextHistoricComData	O	To retrieve the argument of earlier CommissioningErt transactions in ciphertext by an authority.
6.2.13	getCleartextHistoricComData	O	To retrieve the argument of earlier CommissioningErt transactions in cleartext by an ERT holder.
Access control transactions			
6.2.14	updateAccessControlList	O, C	To add or remove authorities or additional service provider to or from the access control list of an ERT.
6.2.15	getCiphertextAccessControlListEntry	O	To retrieve the entries of the access control list in ciphertext by an authority.
6.2.16	getCleartextAccessControlListEntry	O	To retrieve the entries of the access control list in cleartext by the ERT holder.
Miscellaneous			
6.2.17	getErtCapabilities	O	To retrieve the (encryption) capabilities of an ERT.

[a] The column headed "Req" indicates whether a transaction is always required (R), required for confidentiality (C), required for authentication (A) or optional (O).

The definition of the transactions is based on the ASN.1 TRANSACTION information object class as defined in section 6.2.2.

6.2.18 contains definitions common to several transactions.

6.2.2 ERI PDUs and transactions

6.2.2.1 The transaction concept

Writing and reading data to or from an ERT shall be achieved by means of transactions which are defined as instances of the TRANSACTION information object class.

The TRANSACTION information object class is defined as follows:

```
TRANSACTION ::= CLASS {
        &ArgumentType                   ,
        &ResultType                     ,
        &ErrorCodes                     ErrorCode OPTIONAL,
        &transactionCode                INTEGER UNIQUE
        }
WITH SYNTAX {
        ARGUMENT                        &ArgumentType
        RESULT                          &ResultType
        [ERRORS                         &ErrorCodes]
        CODE                            &transactionCode
        }
```

A transaction shall be invoked by an ERI reader or an ERI writer and shall be performed by the ERT.

A transaction is considered atomic in the sense that no data in the ERT will be changed if the transaction does not succeed. See the specification of the individual transactions for details.

No transaction shall be invoked by an onboard ERT reader/writer while the ERT is still performing another transaction. If not the response may be unpredictable, but the atomicity of the transactions shall still be guaranteed.

The &ArgumentType field shall specify the data type of the argument of the transaction.

If a transaction is performed successfully it returns a result as specified by the &ResultType field. If not, it returns an error code as specified by the &ErrorCode field.

The &ResultType field shall specify the data type of the value returned with the result of the transaction when the transaction is successfully performed.

The &ErrorCode field, if present, is of type ErrorCode.

The &transactionCode field specifies the integer value which is used to identify the type of the transaction,.

6.2.2.2 ERI protocol data units

The ERI protocol data units (PDUs) are defined as follows:

```
EriPdu ::= CHOICE {
        requestPdu                  EriRequestPdu,
        reponsePdu                  EriResponsePdu
        }

EriRequestPdu ::= SEQUENCE {
        transactCode                TRANSACTION.&transactionCode ( {EriTransactions}),
        argument                    TRANSACTION.&ArgumentType
                                    ( {EriTransactions} {@.transactCode}) OPTIONAL
        }

EriResponsePdu ::= CHOICE {
        resultPdu                   EriResultPdu,
        errorPdu                    EriErrorPdu
        }

EriResultPdu ::= SEQUENCE {
        transactCode                TRANSACTION.&transactionCode ( {EriTransactions}),
        result                      TRANSACTION.&ResultType ( {EriTransactions} {@.transactCode})
        }

EriErrorPdu ::= SEQUENCE {
        transactCode                TRANSACTION.&transactionCode ( {EriTransactions}),
        error                       TRANSACTION.&ErrorCodes ( {EriTransactions} {@.transactCode})
        }

EriTransactions TRANSACTION ::= {
        getEriData | getAuthenticatedEriData |
        setEriData | getCiphertextHistoricEriData | getCleartextHistoricEriData |
        getPublicCertificateVerificationKeyId | getPublicEnciphermentKeyErt |
        commissionErt | withdrawCommissioning |
        getCiphertextHistoricComData | getCleartextHistoricComData |
        updateAccessControlList | getCiphertextAccessControlListEntry | getCleartextAccessControlListEntry |
        getErtCapabilities
        }
```

An ERI protocol data unit is of type EriData and is either an EriRequestPdu or an EriResponsePdu.

The EriRequestPdu protocol data unit shall be used for the invocation of a transaction to be performed by the ERT.

The EriResponsePdu data protocol unit shall be used for the result or the notification of an error of a transaction performed by the ERT.

EriTransactions specifies the set of ERI transactions (see the clauses below for details).

The transactCode component shall contain the value of the transaction code for a transaction in the set EriTransactions.

The argument component, if present, shall be of the same type as specified for the argument of the transaction identified by the value of transactCode.

The argument component shall be present if the argument for the transaction is defined and absent if not.

The result component shall be of the same type as specified for the result of the transaction identified by the value of transactCode.

The error component shall be an error code from the set of error codes as specified for the transaction identified by the value of transactCode.

6.2.3 Get ERI data

6.2.3.1 The get ERI data transaction

The getEriData transaction shall be used for reading the ERI data with an ERI reader. The getEriData transaction is defined as follows.

```
getEriData TRANSACTION ::= {
        ARGUMENT                GetEriDataArgument
        RESULT                  GetEriDataResult
        ERRORS                  {notCustomized}
        CODE                    1
        }
```

The getEriData transaction is invoked with argument GetEriDataArgument and returns the result GetEriDataResult.

In the case where the ERT is not yet customized, the notCustomized error code shall be returned.

The getEriData transaction is the only time-critical ERI transaction.

NOTE This is the only transaction that has to be performed when the vehicle is driving. Moreover, one should not be able to prevent identification through excessive vehicle speed.

6.2.3.2 The get ERI data transaction argument

The GetEriDataArgument type is defined as follows:

```
GetEriDataArgument ::= SEQUENCE {
        onBehalfOf              EntityId OPTIONAL,
        challenge               Challenge,
        includeAdditionalData   BOOLEAN
        }
```

The onBehalfOf component, if present, specifies the entity whose public encryption key shall eventually be used for the data to be returned. If not present when the ERT is commissioned, the default value is the entityId of the registration authority that commissioned the ERT. When the ERT is not yet commissioned a value of the onBehalfOf component shall be ignored.

NOTE The component onBehalfOf will identify either an authority or an additional service provider.

The challenge component is a random number generated by the ERI reader which shall be returned in the result to show that the result returned is indeed the result of this invocation and not a replicated message.

The includeAdditionalData component shall be used to specify whether or not the result will contain additional vehicle data. If FALSE, only the vehicle identifier is included; if TRUE also the additional vehicle data provided by the vehicle's registration authority, if any, shall also be included.

6.2.3.3 The get ERI data transaction result

The GetEriDataResult type is defined as follows:

```
GetEriDataResult ::= SEQUENCE {
        registrationAuthority   EntityId OPTIONAL,
        eriResultData           SECURED {CleartextEriData}}
        }
```

The registrationAuthority component, if present, shall identify the vehicle's registration authority.

The registrationAuthority component shall be present whenever the ERT is commissioned and absent if not.

The eriResultData component shall contain the secured cleartext ERI data (see 6.2.18.1).

The cleartext ERI data shall be secured, i.e. signed and/or encrypted, in accordance with capabilities of the ERT.

6.2.3.4 Cleartext ERI data

The CleartextEriData type is used for ERI data as requested by an invocation of a getEriData or GetAuthenticatedEriData transaction and is defined as follows:

```
CleartextEriData ::= SEQUENCE {
        eriDataOrId                 EriDataOrId,
        ErtSecurityStatus           ErtSecurityFlags OPTIONAL
        }
```

The eriDataOrId component is of type EriDataOrId as defined below.

The ertSecurityStatus component, if present, shall be of type ErtSecurityFlags as defined below.

The ertSecurityStatus component shall be present if the ERT has the capability to set or reset at least one of the security status bits. If not, the component shall not be present.

6.2.3.5 ERI data or id

The EriDataOrId type is used to specify either the VehicleId or the ERI data and is defined as follows:

```
EriDataOrId ::= CHOICE {
        vehicleId                   VehicleId,
        unsignedDatedEriData        DATED {EriData},
        datedAndSignedEriData       SIGNED {DATED {EriData}, PrivateSignatureKey} -- BOE signed
        }
```

The vehicleId alternative shall be used when the includeAdditionalData component in the argument field of the invoked transaction has the value FALSE.

The unsignedDatedEriData component shall be used when the includeAdditionalData component in the argument field of the invoked transaction has the value TRUE and the ERI data was not signed when written into the ERT.

The value of the unsignedDatedEriData component shall be the value of the dated ERI data as written into the ERT by the last set ERI data transaction.

The datedAndSignedEriData component shall be used when the includeAdditionalData component in the argument field of the invoked transaction has the value TRUE and the ERI data was signed when written into the ERT.

The value of the datedAndSignedEriData component shall be the value of the dated and signed ERI data as written into the ERT by the last successfully performed set ERI data transaction.

6.2.3.6 ERI data

The EriData type shall be used for the ERI data and is defined as follows:

```
EriData ::= SEQUENCE {
        id                      VehicleId,
        additionalEriData       OCTET STRING (CONTAINING AdditionalEriData) OPTIONAL
        }
```

The id component shall contain the vehicle id as defined in ISO 24534-3.

The additionalEriData component shall be an octet string containing the additional ERI data as defined in ISO 24534-3.

NOTE By wrapping the additional ERI data into an octet string the ERT does not have to code or decode the additional ERI data. Therefore, future changes of the definition of the additional ERI data type will have no impact on the design of the ERT too (apart from the maximum size of an ERI data record).

6.2.3.7 ERT security flags

The purpose of the ERT security flags is to signal security treats that may e.g. be caused by attempts to tamper with the ERT.

NOTE 1 The security capabilities of the ERT can be obtained with the get ERT capabilities transactions.

The ErtSecurityFlags type is used to specify the security flags and is defined as follows:

```
ErtSecurityFlags ::= BIT STRING {
        flagsHaveBeenResetted (0),
        notCommissioned (1),
        lowSupplyVoltageIndication (2),
        highSupplyVoltageIndication (3),
        lowClockFrequencyIndication (4),
        highClockFrequencyIndication (5),
        lowTemperatureIndication (6),
        highTemperatureIndication (7)
        } (SIZE (0..16))                -- bit 8 .. 15 reserved for future use
```

The initial value of the bit string shall be '0100000000000000'B, i.e. not yet commissioned, nothing yet detected and no reset.

If the value of an ErtSecurityFlags type is '0000000000000000'B or '0100000000000000'B a reset operation will not change this value. If not, a reset operation will set it to '1000000000000000'B.

The notCommissioned bit will be set to 1 if a commissioning transaction is performed successfully and set to 0 if a withdraw commissioning transaction is invoked.

NOTE 2 The value of an ErtSecurityFlags component can be reset with a commission ERT transaction.

Bit 2 .. 15 will be set to 1 if the corresponding condition has been detected.

NOTE 3 A value 0 only indicates that the condition has not been detected. This may be because an ERT does not have the corresponding detecting capability. See also 6.2.17.

6.2.4 Authenticated ERI data

6.2.4.1 The authenticated ERI data transaction

The getAuthenticatedEriData transaction shall be used to obtain an authenticated version of the ERI data and is defined as follows:

```
getAuthenticatedEriData TRANSACTION  ::=  {
        ARGUMENT                        GetAuthenticatedEriDataArgument
        RESULT                          GetAuthenticatedEriDataResult
        ERRORS                          {notCustomized}
        CODE                            2
        }
```

The getAuthenticatedEriData transaction is invoked with argument GetAuthenticatedEriDataArgument.

If the ERT is customized the GetAuthenticatedEriDataResult is returned.

If the ERT is not customized the nonCustomized error code is returned.

6.2.4.2 The authenticated ERI data transaction argument

The GetAuthenticatedEriDataArgument type is defined as follows:

```
GetAuthenticatedEriDataArgument ::= SEQUENCE {
        ertHolderCredentials            ErtHolderCredentials,
        challenge                       Challenge,
        includeAdditionalData           BOOLEAN
        }
```

The ertHolderCredentials component shall contain the ERT holder's credentials, i.e. the vehicleId and the ERT holder's PIN.

The challenge component and includeAdditionalData component are defined as in the GetEriDataArgument type.

6.2.4.3 The authenticated ERI data transaction result

The GetAuthenticatedEriDataResult type is defined as follows:

```
GetAuthenticatedEriDataResult ::= SEQUENCE {
        registrationAuthority           EntityId OPTIONAL,
        authenticateResultData          CLEAR-SECURED {CleartextEriData}
        }
```

The registrationAuthority component, if present, shall identify the vehicle's registration authority.

The registrationAuthority component shall be present if the ERT is commissioned and absent if not.

The authenticateResultData component shall contain the secured but not encrypted cleartext ERI data (see 6.2.18.1).

The cleartext ERI data is clear secured, i.e. signed or not, in accordance with capabilities of the ERT.

6.2.5 Set ERI data

6.2.5.1 The set ERI data transaction

The setEriData transaction shall be used for customizing and updating the ERT and is defined as follows:

```
setEriData TRANSACTION ::= {
        ARGUMENT                SetEriDataArgument
        RESULT                  NULL
        ERRORS                  {SetEriDataErrors}
        CODE                    3
        }
```

The setEriData transaction is invoked with an argument of type SetEriDataArgument.

If the transaction is performed successfully, the ERT returns a NULL value. If not, one of the error codes from SetEriDataErrors is returned.

No data shall be stored in the ERT if the transaction is not successfully performed.

When the transaction is performed successfully for the first time by an ERT, the ERT is said to be customized. When an ERT is customized the vehicleId in this ERT can never be changed.

NOTE If the ERT does not have signature verification capabilities, any writer can write any ERI data into the ERT.

6.2.5.2 The set ERI data transaction argument

The SetEriDataArgument type is defined as follows:

```
SetEriDataArgument ::= CHOICE {
        clearTextArgument       ClearTextSetEriDataArgument,
        encryptedArgument       ENCRYPTED {ClearTextSetEriDataArgument}
        }
```

The encryptedArgument alternative shall not be chosen if the ERT has no decipherment capabilities.

NOTE Whether or not an ERT has decipherment capabilities can be determined with the get ERT capabilities transaction.

If the encryptedArgument alternative has been chosen, the ClearTextSetEriDataArgument value shall be encrypted with the public encipherment key that can be obtained with the getPublicEnciphermentKeyErt transaction.

6.2.5.3 The cleartext set ERI data argument type

The ClearTextSetEriDataArgument type is defined as follows:

```
ClearTextSetEriDataArgument ::= CHOICE {
        authenticatedEriData    BOE-AUTHENTICATED {DATED {EriData}},
        notAuthenticatedEriData DATED {EriData}
        }
```

The authenticatedEriData alternative shall be chosen by the ERI writer whenever the ERT has signature verification capabilities and may be chosen when the ERT has no signature verification capabilities. The notAuthenticatedEriData may only be chosen if the ERT has no signature verification capabilities.

NOTE Whether or not an ERT has signature verification capabilities can be determined with the get ERT capabilities transaction.

The authenticatedEriData, if present, shall contain the ERI data as dated and signed (see 6.2.18.3.1 and 6.2.18.4) by the registration authority or manufacturer including the certificate(s) for the public verification key of the registration authority or manufacturer.

The notAuthenticatedEriData, if present, shall contain the ERI data as dated (see 6.2.18.3.1) by the registration authority.

When the transaction is invoked for a second or subsequent time, the vehicleId in the argument shall be the same as when the ERT was customized.

6.2.5.4 Logging of set ERI data arguments

The ClearTextSetEriDataArgument value of each setEriData transaction shall be sequentially numbered and stored in the ERT for later retrieval. The ClearTextSetEriDataArgument value of the customization transaction shall be numbered 1.

The ERT shall be capable of storing one or more ClearTextSetEriDataArgument values, i.e. zero or more values from successfully performed previous set ERI data invocations.

If the maximum number of ClearTextSetEriDataArgument values that the ERT can store has been reached, the following procedure shall be applied:

— In the case where the ERT can store only one ClearTextSetEriDataArgument value, i.e. no historic data, the new ClearTextSetEriDataArgument value replaces the old one.

— In the case where the ERT can store two or more ClearTextSetEriDataArgument values, i.e. can store historic data, the new ClearTextSetEriDataArgument value replaces the second oldest present ClearTextSetEriDataArgument value.

NOTE By removing the second oldest and not the oldest value, the original ERI data is retained.

If the size of the ClearTextSetEriDataArgument value exceeds the maximum value allowed for this argument, the resourceLimitExceeded error code is returned and there is no update of the ERI data.

6.2.5.5 The authentication of ERI data

No authentication data shall have a date component with a date earlier than that of a previous setEriData invocation.

NOTE This is also a protection against a replay attack. It prevents a non-authorized ERI writer from illegally reinstalling a historic version of the ERI data.

If the transaction is invoked for a second or subsequent time while the ERT is not yet commissioned, the value of the issuer component in the dated data shall be the same as in the customizing transaction (i.e. from the same manufacturer or registration authority).

If the transaction is invoked when the ERT is commissioned, the entityId in the authentication data shall then be the same as in the commissioning transaction (i.e. from the same registration authority).

The transaction shall not be invoked while the commissioning is withdrawn.

In the cases where the ERT has signature verification capabilities, the following shall apply:

— When an ERT has never been commissioned, the data in the argument shall be signed by a manufacturer or a registration authority.

— When an ERT has been commissioned at least once, the data in the argument shall be signed by a registration authority.

— Access to the ERT shall be controlled by certificates included in the argument. All certificates in the argument shall be valid (i.e. already issued and not yet expired) at the date contained in the date component in the authentication data.

6.2.5.6 The set ERI data transaction errors

The SetEriDataErrors type is a subtype of the ErrorCode type defined as follows:

```
SetEriDataErrors ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalCertificate |            illegalSignature |
        illegalDate |                   notCommissioned |
        resourceLimitExceeded |         otherError
        }
```

The SetEriDataErrors are of type EriErrorCode and comprise the listed values.

When the ERT has signature verification capabilities and the ERI data in the argument is not authenticated, the illegalArgument value shall be returned.

When the transaction is invoked while the commissioning is withdrawn, the notCommissioned value shall be returned.

6.2.6 Get ciphertext historic ERI data

6.2.6.1 The get ciphertext historic ERI data transaction

The getCiphertextHistoricEriData transaction shall be used to retrieve the eventually encrypted argument of a previous setEriData transaction and is defined as follows:

```
getCiphertextHistoricEriData TRANSACTION ::= {
        ARGUMENT                GetCiphertextHistoricEriDataArgument
        RESULT                  SECURED {HistoricEriData}
        ERRORS                  {notCustomized}
        CODE                    4
        }
```

The getCiphertextHistoricEriData transaction is invoked with argument GetCiphertextHistoricEriDataArgument.

If the ERT is customized, the transaction returns the historic ERI data (see below) signed and/or encrypted according to the capabilities of the ERT. See 6.2.18.1 for the definition of SECURED {}.

In case the ERT is not yet customized, the nonCustomized error code is returned.

6.2.6.2 The get ciphertext historic ERI data transaction argument

The GetCiphertextHistoricEriDataArgument type is defined as follows:

```
GetCiphertextHistoricEriDataArgument ::= SEQUENCE {
        onBehalfOf              EntityId OPTIONAL,
        challenge               Challenge,
        number                  INTEGER (1..int4)
        }
```

The onBehalfOf component, if present, specifies the entity whose public encryption key shall eventually be used for the data to be returned. If not present, the default value is the entityId of the vehicle's registration authority. When the ERT is not yet commissioned, a value of the onBehalfOf component shall be ignored.

The challenge component is a random number generated by the ERI reader which shall eventually be returned in the signed part of the result to show that the result returned is the result of this invocation and not a replicated message.

The stored arguments of setEriData transactions are sequentially numbered. The first stored argument (of the customizing setEriData transaction) is numbered 1.

The number component identifies the argument of a setEriData invocation.

When the value of the number component is the number of a deleted argument, the oldest available argument before the deleted one of a setEriData transaction shall be returned.

When the value of the number component is greater than the number of the most recent argument stored, the data of this most recent argument shall be returned.

6.2.6.3 Historic ERI data

The HistoricEriData type is used for the retrieval of arguments and to set ERI data transactions and is defined as follows:

```
HistoricEriData ::= SEQUENCE {
        number                  INTEGER (1..int4),
        outOf                   INTEGER (1..int4),
        historicRecord          ClearTextSetEriDataArgument
        }
```

The number component of the HistoricEriData identifies the ClearTextSetEriDataArgument value conveyed in the historicRecord component.

The outOf component specifies the number of the most recently stored ClearTextSetEriDataArgument value.

The historyRecord component contains the ClearTextSetEriDataArgument value identified by the value of the number component.

6.2.7 Get cleartext historic ERI data

6.2.7.1 The get cleartext historic ERI data transaction

The getCleartextHistoricEriData transaction shall be used to retrieve the argument of previous setEriData transactions in cleartext and is defined as follows:

```
getCleartextHistoricEriData TRANSACTION ::= {
        ARGUMENT                GetCleartextHistoricEriDataArgument
        RESULT                  CLEAR-SECURED {HistoricEriData}
        ERRORS                  {notCustomized}
        CODE                    5
        }
```

The ARGUMENT field is of the type GetCleartextHistoricEriDataArgument.

The transaction returns the historic ERI data (see 6.2.6.3) signed or not according to the capabilities of the ERT. See 6.2.18.1 for the definition of CLEAR-SECURED {}.

In case the ERT is not yet customized, the nonCustomized error code is returned.

6.2.7.2 The get cleartext historic ERI data transaction argument

The GetCleartextHistoricEriDataArgument type is defined as follows:

```
GetCleartextHistoricEriDataArgument ::= SEQUENCE {
        credentials                 ErtHolderCredentials,
        challenge                   Challenge,
        number                      INTEGER (1..int4)
        }
```

The credentials component is defined as in the argument of the getAuthenticatedEriData transaction.

The challenge and number components are defined as in the argument of the getCiphertextHistoricEriData transaction.

6.2.8 Get public certificate verification key identifier transaction

The getPublicCertificateVerificationKeyId transaction is used to retrieve the identifier of the public verification key to be used for the verification of top-level public key certificates. The getPublicCertificateVerificationKeyId transaction is defined as follows:

```
getPublicCertificateVerificationKeyId TRANSACTION ::= {
        ARGUMENT                    NULL
        RESULT                      KeyId
        CODE                        6
        }
```

The argument field is of type NULL.

The result of the transaction is the identifier of the public verification key to be used for the verification of top-level public key certificates. If the ERT does not support signature verification, or this key is not available, the value returned shall be zero.

NOTE This allows the top-level certification authority to use multiple key pairs for its certificates. The key identifier can then be used to select the right certificate(s) for a write transaction.

6.2.9 Get public encipherment key ERT

6.2.9.1 The get public encipherment key ERT transaction

The getPublicEnciphermentKeyErt transaction shall be used to retrieve the public encipherment key of the ERT by a registration authority. This key is needed for the encryption of private commissioning data. The getPublicEnciphermentKeyErt transaction is defined as follows:

```
getPublicEnciphermentKeyErt TRANSACTION ::= {
        ARGUMENT                    BOE-AUTHENTICATED {vehicleId}
        RESULT                      PublicEnciphermentKey
        ERRORS                      {GetPublicEnciphermentKeyErrors}
        CODE                        6
        }
```

The argument field is of type BOE-AUTHENTICATED {vehicleId} and contains the vehicleId as authenticated (see 6.2.18.4) by the BOE. The signatory of the authentication data shall be a registration authority.

NOTE As the public encipherment key of the ERT is used for the encryption of other keys, it is a master key that should not be made available to entities without real need.

If successful, the transaction returns the public encipherment key for the ERT. If not successful, the transaction returns a GetPublicEnciphermentKeyErrors value as defined below.

6.2.9.2 The get public encipherment key ERT transaction errors

The GetPublicEnciphermentKeyErrors type is a subtype of the ErrorCode type defined as follows:

```
GetPublicEnciphermentKeyErrors  ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalCertificate |            illegalSignature |
        illegalEntity |                 noDeciphermentCapability |
        otherError                      }
```

The GetPublicEnciphermentKeyErrors type is of type EriErrorCode and shall take one of the listed values.

6.2.10 Commission an ERT

6.2.10.1 The commission ERT transaction

A registration authority shall use the commissionErt transaction to commission an ERT or to update the ERT security parameters. The commissionErt transaction is defined as follows:

```
commissionErt TRANSACTION ::= {
        ARGUMENT                        CommissionErtArgument
        RESULT                          NULL
        ERRORS                          { CommissionErtErrors }
        CODE                            7
        }
```

NOTE 1 A commissioningErt transaction may only be invoked when the ERT is customized.

The commissionErt transaction argument is of type CommissionErtArgument.

If the transaction is performed successfully, the ERT returns a NULL value. If not, one of the error codes from CommissionErtErrors is returned.

No data shall be stored in the ERT if the transaction is not successfully performed.

NOTE 2 If the ERT does not have signature verification capabilities, any writer can write any commissioning data into the ERT.

6.2.10.2 The commissioning ERT argument

The CommissionErtArgument type is defined as follows:

```
CommissionErtArgument ::= CHOICE {
        authenticatedData               BOE-AUTHENTICATED { DATED {CommissioningData}},
        notAuthenticatedData            DATED {CommissioningData}
        }
```

The authenticatedData alternative shall be chosen whenever the ERT has signature verification capabilities and may be chosen when the ERT has no signature verification capabilities. The notAuthenticatedData may only be chosen if the ERT has no signature verification capabilities.

The authenticatedData, if present, shall contain the commissioning data as dated and signed (see 6.2.18.3.1 and 6.2.18.4) by the registration authority including the certificate(s) for the public verification key of the registration authority.

The notAuthenticatedData, if present, shall contain the commissioning data as dated (see 6.2.18.3.1) by the registration authority.

6.2.10.3 Logging of commissioning ERT and withdraw commissioning arguments

The arguments of commission ERT and withdraw commissioning transactions that are performed successfully shall be sequentially numbered and stored for later retrieval. The argument of the first successful invocation shall be numbered 1.

The ERT shall be capable of storing one or more commission ERT and withdraw commissioning transaction arguments, i.e. zero or more arguments of successfully performed previous commission ERT or withdraw commissioning transaction invocations.

If the maximum number of commission ERT and withdraw commissioning transaction arguments that the ERT can store has been reached, the following procedure shall be applied:

— In case the ERT can store only one argument, i.e. no historic data, the new argument replaces the old one.

— In case the ERT can store two or more arguments, i.e. can store historic data, the new argument replaces the second oldest present argument.

If the size of the argument exceeds the maximum size the ERT can store, the resourceLimitExceeded error code is returned.

6.2.10.4 The commissioning data

The commissioningData type contains all the data required for commissioning the ERT and is defined as follows:

```
CommissioningData ::= SEQUENCE {
        vehicleId                       VehicleId,
        registrationAuthority           EntityId,
        resetSecurityFlags              BOOLEAN,
        enciphermentKeyId               KeyId OPTIONAL,
        publicEnciphermentKeyAuthority  PublicEnciphermentKey OPTIONAL,
        publicVerificationKeyCertificate ErtCertificate OPTIONAL,
        privateData                     ENCRYPTED {PrivateCommissioningData} OPTIONAL
        }
```

The vehicleId component identifies the vehicle. The value shall be the same as used for the customization of the ERT.

The registrationAuthority component identifies the registration authority that (re)commissioned the ERT.

The resetSecurityFlags component shall specify whether or not the security flags shall be reset. If TRUE, they shall be reset; if FALSE, they shall not be reset.

The enciphermentKeyId component, if present, identifies the public encipherment key used for the encryption. Different registration authorities may use the same identifiers.

NOTE 1 When returned with encrypted data this value can be used to determine the corresponding private decipherment key.

The enciphermentKeyId component shall be present if the publicEnciphermentKeyAuthority component is present. If not, the component shall be absent.

The publicEnciphermentKeyAuthority component, if present, contains the public encipherment key of the registration authority identified by the registrationAuthority component. This key shall be used for the encryption of results returned by the ERT, if applicable.

The publicEnciphermentKeyAuthority component shall be present when the ERT has encipherment capabilities that are to be enabled. If not, the component shall be absent and any ERT encipherment capabilities, if present, shall be disabled.

NOTE 2 This brings the use of any encipherment capabilities of the ERT, if present, under control of the commissioning registration authority.

The publicVerificationKeyCertificate component, if present, shall contain the certificate for the public verification key for the verification of ERT signatures. This certificate shall be issued by the registration authority as identified in the registrationAuthority component.

The publicVerificationKeyCertificate component shall be present if the privateSignatureKeyErt component is present in the privateData component. If not, it shall be absent.

The privateData component, if present, contains the encrypted PrivateCommssioningData. The public encryption key used shall be the PublicEnciphermentKey as can be obtained with a getPublicEnciphermentKeyErt transaction.

The privateData component shall be present if the PrivateCommissioningData type contains at least one component. If not, it shall be absent.

6.2.10.5 The private commissioning data

The PrivateCommissioningData type contains all the private data required for commissioning the ERT and is defined as follows:

```
PrivateCommissioningData ::= SEQUENCE {
        privateSignatureKeyErt      PrivateSignatureKey OPTIONAL,
        pin                         PIN OPTIONAL
        }
```

The privateSignatureKeyErt component, if present, contains the private signature key to be used by the ERT to sign the result of transactions.

The privateSignatureKeyErt component shall be present when the ERT has signing capabilities that are to be enabled. If not, the component shall be absent and any signing capabilities, if present, shall be disabled.

NOTE 1 A key identifier is not required. An ERT always attaches a certificate for the corresponding public verification key to each ERT signature.

The pin component, if present, shall contain a random PIN that can be used in conjunction with the vehicleId to get access to cleartext ERI data (e.g. by the ERT holder).

NOTE 2 The secure distribution of a PIN to an ERT holder is outside the scope of this part of ISO 24534.

The pin component shall be present if the encipherment capabilities of the ERT are enabled and the getAuthenticatedEriData, getCleartextHistoricEriData, UpdateAccesControlList (for holders) or getCleartext-AccessControlListEntry should be supported. If not, it should be absent.

In case the pin component is present while the encipherment capabilities of the ERT are disabled, its value shall be ignored.

6.2.10.6 The commission ERT transaction errors

The CommissionErtErrors type is defined as follows:

```
CommissionErtErrors ErrorCode ::= {
        illegalArgument |                   illegalVehicleId |
        illegalCertificate |                illegalSignature |
        illegalEntity |                     illegalDate |
        notCustomized |                     resourceLimitExceeded |
        noEnciphermentCapability |
        secretKeyEncryptionAlgorithmNotSupported |
        publicKeyEncryptionAlgorithmNotSupported |
        noSigningCapability |
        hashingAlgorithmNotSupported |
        signingAlgorithmNotSupported |
        otherError                          }
```

The CommissionErtErrors type is of type ErrorCode and shall take one of the listed values.

6.2.11 Withdraw commissioning

6.2.11.1 The withdraw commissioning transaction

The withdrawCommissioning transaction shall be used to withdraw the commissioning of an ERT, e.g. in the case of the end of life of the vehicle.

The withdrawCommissioning transaction is defined as follows:

```
withdrawCommissioning TRANSACTION ::= {
        -- this transaction also removes all public encipherment keys
        ARGUMENT                    WithdrawCommissioningArgument
        RESULT                      SECURED {WithdrawCommissioningResultData}
        ERRORS                      {WithdrawCommissioningErrors}
        CODE                        withdrawCommissioningCode
        }

withdrawCommissioningCode INTEGER ::= 9
```

The ARGUMENT field shall be of type WithdrawCommissioningArgument.

If the ERT is commissioned, the transaction returns the SECURED WithdrawCommissioningResultData (see below) signed and/or encrypted according to the capabilities of the ERT (see 6.2.18.1 for the definition of SECURED {}). If not, one of the error codes from CommissionErtErrors is returned.

The transaction may only be invoked by the registration authority that has commissioned the ERT.

Performing this transaction will result in:

— removal of all keys and PIN, if any, set by the registration authority by means of a commissioning transaction,

— removal of all entries from the access control list,

— disabling, if applicable, of the encipherment capabilities of the ERT,

— setting of the “notCommissioned” bit in the ERT security flags, and

— numbering and storage of the argument value (see 6.2.10.3).

NOTE The basic idea of this transaction it to disable further use of the ERT. However, in order to prevent illegal use of the vehicle after this transaction, its performance is only signalled (in the ERT security flags) and the ERT is not really disabled. And, because there is no reason to protect the privacy of something that should not be used at all, all encryption capabilities are also disabled.

6.2.11.2 The withdraw commissioning transaction argument

The WithdrawCommissioningArgument type is defined as follows:

```
WithdrawCommissioningArgument ::= CHOICE {
        authenticatedData           BOE-AUTHENTICATED {VehicleId},
        notAuthenticatedData        VehicleId
        }
```

The authenticatedData alternative shall be chosen whenever the ERT has signature verification capabilities, and may be chosen when the ERT has no signature verification capabilities. The notAuthenticatedData may only be chosen if the ERT has no signature verification capabilities.

The authenticatedData, if present, shall contain the vehicle identifier as signed (see 6.2.18.4) by the registration authority including the certificate(s) for the public verification key of the registration authority.

The notAuthenticatedData, if present, shall contain the vehicle identifier.

NOTE In case the ERT has no signature verification capabilities, everyone can withdraw the commissioning even if the authenticatedData alternative is chosen (if a signature is not verified any dummy signature can be used).

6.2.11.3 The withdraw commissioning transaction result data

The WithdrawCommissioningResult type is defined as follows:

```
WithdrawCommissioningResultData ::= [APPLICATION withdrawCommissioningCode ] SEQUENCE {
        withdrawn                   WithdrawCommissioningArgument,
        historicComData             HistoricComData
        }
```

The withdrawn component shall be used to return the argument of the transaction.

The historicComData component shall contain the value of the most recently stored argument of a commissioning ERT transaction.

6.2.11.4 The withdraw commissioning transaction errors

The WithdrawCommissioningErrors type is defined as follows:

```
WithdrawCommissioningErrors ErrorCode ::= {
        illegalArgument |           illegalVehicleId |
        illegalCertificate |        illegalSignature |
        illegalEntity |             illegalDate |
        notCustomized |             notCommissioned |
        otherError                  }
```

The WithdrawCommissioningErrors type is of type EriErrorCode and shall take one of the listed values.

6.2.12 Get ciphertext historic commissioning data

6.2.12.1 The get ciphertext historic commissioning data transaction

The getCiphertextHistoricComData transaction shall be used to retrieve the eventually encrypted argument of a previous commissionErt transaction and is defined as follows:

```
getCiphertextHistoricComData TRANSACTION ::= {
        ARGUMENT                    GetCiphertextHistoricComDataArgument
        RESULT                      SECURED {HistoricComData}
        ERRORS                      {notCommissioned}
        CODE                        12
        }
```

The ARGUMENT field shall be of type GetCiphertextHistoricComDataArgument.

If the ERT was or is commissioned, the transaction returns the historic commissioning data (see below) signed and/or encrypted according to the capabilities of the ERT (see 6.2.18.1 for the definition of SECURED {}).

When the ERT was never commissioned, the notCommissioned error code shall be returned.

6.2.12.2 The get ciphertext historic commissioning data transaction argument

The GetCiphertextHistoricEriDataArgument type is defined as follows:

```
GetCiphertextHistoricComDataArgument ::= SEQUENCE {
        onBehalfOf                  EntityId OPTIONAL,
        challenge                   Challenge,
        number                      INTEGER (1..int4)
        }
```

The onBehalfOf component, if present, specifies the entity whose public encryption key shall eventually be used for the data to be returned. If not present, the default value is the entityId of the vehicle's registration authority. When the ERT is not yet commissioned, a value of the onBehalfOf component shall be ignored.

The challenge component is a random number generated by the ERI reader which shall eventually be returned in the signed part of the result to show that the result returned is the result of this invocation and not a replicated message.

The stored arguments of commissionErt transactions are sequentially numbered. The first stored argument is numbered 1.

The number component identifies the argument of a commissionErt invocation.

When the value of the number component is the number of a deleted argument, the oldest available argument of a successfully invoked commissionErt transaction shall be returned.

When the value of the number component is greater than the number of the most recent argument stored, the data of this most recent argument shall be returned.

6.2.12.3 The historic commissioning data

The HistoricComData type shall be used for historic commissioning data and is defined as follows:

```
HistoricComData ::= SEQUENCE {
        number                  INTEGER (1..int4),
        outOf                   INTEGER (1..int4),
        historicRecord          CommissionErtArgument
        }
```

The number component identifies the argument of a commissionErt transaction conveyed in the historicRecord component.

The outOf component specifies the number of the most recently stored argument of a commissionErt transaction.

In case the ERT only contains the current commissioning data and no historic data, the value of both the number and out-of-number components shall be 1.

The historyRecord component contains the argument of the successful commissionErt transaction invocation identified by the value of the number component.

6.2.13 Get cleartext historic commissioning data

6.2.13.1 The get cleartext historic commissioning data transaction

The getCleartextHistoricComData transaction shall be used to retrieve the argument of previous commissionErt transactions in cleartext and is defined as follows:

```
getCleartextHistoricComData TRANSACTION ::= {
        ARGUMENT                GetCleartextHistoricComDataArgument
        RESULT                  CLEAR-SECURED {HistoricComData}
        ERRORS                  {notCommissioned}
        CODE                    13
        }
```

The ARGUMENT field is of the type GetCleartextHistoricComDataArgument.

The transaction returns the historic commissioning data (see below) signed or not according to the capabilities of the ERT (see 6.2.18.1 for the definition of CLEAR-SECURED {}).

In case the ERT is not yet commissioned or if the commissioning is withdrawn, the notCommissioned error code is returned.

6.2.13.2 The get cleartext historic commissioning data transaction argument

The GetCleartextHistoricEriDataArgument type is defined as follows:

```
GetCleartextHistoricComDataArgument ::= SEQUENCE {
        credentials             ErtHolderCredentials,
        challenge               Challenge,
        number                  INTEGER (1..int4)
        }
```

The components of the GetCleartextHistoricComDataArgument are defined as in the GetCleartextHistoric-EriDataArgument.

6.2.14 Update of the access control list

6.2.14.1 The update access control list transaction

The updateAccessControlList transaction shall be used to add or delete entries to the access control list of the ERT. The updateAccessControlList transaction is defined as follows:

```
updateAccessControlList TRANSACTION ::= {
        ARGUMENT                    UpdateAccessControlListArgument
        RESULT                      NULL
        ERRORS                      {UpdateAccessControlListErrors}
        CODE                        11
        }
```

The updateAccessControlList transaction is invoked with an argument of type updateAccessControlList-Argument.

The transaction may be invoked either by the registration authority who commissioned the ERT or the ERT holder.

If the transaction is performed successfully, the ERT returns a NULL value. If not, one of the error codes from UpdateAccessControlListErrors is returned.

No data shall be stored in the ERT if the transaction is not successfully performed.

6.2.14.2 The update access control list argument

The UpdateAccessControlListArgument type is defined as follows:

```
UpdateAccessControlListArgument ::= CHOICE {
        authorityUpdate             SIGNED {AccessControlListUpdate},
        holderUpdate                HolderAccessControlListUpdate
        }
```

The authorityUpdate alternative shall be used when the access control list of the ERT is updated by a registration authority. The holderUpdate alternative shall be used when the ERT holder updates the access control list.

The authorityUpdate component, if present, shall have the AccessControlListUpdate data signed with the same key as used for the most recently successfully performed commissioning transaction. If not, an illegalSignature error code shall be returned.

When the ERT has signature verification capabilities, the signature of an authorityUpdate component shall be checked with the certificates included in the argument of the most recent successfully performed commissioning transaction.

6.2.14.3 The holder access control list update type

The HolderAccessControlListUpdate type is defined as follows:

```
HolderAccessControlListUpdate ::= SEQUENCE {
        credentials                 ErtHolderCredentials,
        entry                       AccessControlListUpdate
        }
```

The credentials component shall contain the credentials (vehicleId plus PIN) of the ERT holder. The credentials shall be checked by the ERT.

6.2.14.4 The access control list update type

The AccessControlListUpdate type shall be used to specify the update of the access control list of the ERT and is defined as follows:

```
AccessControlListUpdate ::= SEQUENCE {
        mode                    ENUMERATED {deleteAllAndAdd (0), add (1), delete (2)}
                                DEFAULT add,
        entry                   AccessControlEntry OPTIONAL
        }
```

The mode component specifies the update mode and can have one of the following values:

— deleteAllAndAdd, in which case first either all entries added by a registration authority are deleted if the transaction is invoked by a registration authority or all entries added by an ERT holder are deleted if the transaction is invoked by an ERT holder. Second, a new entry, if present, is added.

— add, to add a new entry to the access control list. Entries added by a registration authority are added as authorities. Entries added by the ERT holder are added as additional service providers.

— delete, to delete an entry from the access control list. A registration authority may only delete entries added by a (possibly other) registration authority. An ERT holder may only delete entries added by a (possibly previous) ERT holder.

The entry component, if present, shall contain the entry to be added or deleted.

The entry component shall be absent if the mode component has the value deleteAllAndAdd and no entries are added. If not, the entry component shall be present.

In case the mode is “add” and if there is already an entry for an entity with the same entityId present, the entry to be added shall completely replace the old one.

6.2.14.5 The access control entry type

The AccessControlEntry type is defined as follows:

```
AccessControlEntry ::= SEQUENCE {
        id                              EntityId,
        name                            Text OPTIONAL,
        enciphermentKeyId               KeyId OPTIONAL,
        publicEnciphermentKeyReader     PublicEnciphermentKey OPTIONAL
        }
```

The id component shall identify the entity whose access right is being added or removed.

The name component, if present, shall contain the name of the entity whose access right is being added or removed. If an entry is to be removed, any value present shall be ignored by the ERT.

The enciphermentKeyId component, if present, identifies the public encipherment key used for the encryption. Different entities may use the same identifiers.

NOTE When returned with encrypted data, this value can be used to determine the corresponding private decipherment key.

The enciphermentKeyId component shall be present if the publicEnciphermentKeyReader component is present. If not, the component shall be absent.

The publicEnciphermentKeyReader component, if present, shall contain the public encipherment key of the entity whose access right is being added. This key shall be used to encrypt ERI data before it is sent as ciphertext to an ERI reader acting on behalf of the entity identified by the id component.

The publicEnciphermentKeyReader component shall be absent if the value of the mode component is “delete”. If not, it shall be present.

6.2.14.6 The update access control list transaction errors

The set of values UpdateAccessControlListErrors is defined as follows:

```
UpdateAccessControlListErrors ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalSignature |              illegalHolderAccess |
        illegalDate |                   noEntry |
        resourceLimitExceeded |         noEnciphermentCapability |
        otherError                      }
```

The values of UpdateAccessControlListErrors are of type EriErrorCode and comprise the listed values.

6.2.15 Get ciphertext access control list entry

6.2.15.1 The get ciphertext access control list transaction

The getCiphertextAccessControlListEntry transaction shall be used to retrieve an access control list entry resulting from an authority update of this list in ciphertext from the ERT and is defined as follows:

```
getCiphertextAccessControlListEntry TRANSACTION ::= {
        ARGUMENT                        GetCiphertextAccessControlListEntryArgument
        RESULT                          SECURED {AuthorityAccessControlListEntry}
        CODE                            9
        }
```

The ARGUMENT field is of type GetCiphertextAccessControlListEntryArgument.

The transaction returns the AuthorityAccessControlListEntry data (see below) signed and/or encrypted according to the capabilities of the ERT (see 6.2.18.1 for the definition of SECURED {}).

If encrypted, the public encipherment key used is the public encipherment key of the registration authority that has commissioned the ERT.

NOTE The identifier of the registration authority that commissioned the ERT can be obtained with a getEriData transaction.

The getCiphertextAccessControlListEntry transaction operates on an access control list. This access control list is the list of all access control entries in the ERT that were added or updated by means of an updateAccessControlList transaction with an authorityUpdate argument. Each entry in this list, if any, is given a distinct number between 1 and the number of entries in the list.

The access control list may be empty.

6.2.15.2 The get ciphertext access control list entry transaction argument

The type GetCiphertextAccessControlListEntryArgument is defined as follows:

```
GetCiphertextAccessControlListEntryArgument ::= SEQUENCE {
        challenge                       Challenge,
        number                          INTEGER (1..int4)
        }
```

The challenge component is a random number generated by the ERI reader which shall eventually be returned in the signed part of the result to show that the result returned is the result of this invocation and not a replicated message.

The number component specifies the number of the entry to be retrieved from the access control list the transaction is operating on.

6.2.15.3 The authority access control list entry

The AuthorityAccessControlListEntry type and AccessControlListEntry type are defined as follows:

```
AuthorityAccessControlListEntry ::= AccessControlListEntry (WITH COMPONENTS {…, holderEntry ABSENT} )

AccessControlListEntry ::= SEQUENCE {
        number                          INTEGER (0..int4),
        outOf                           INTEGER (0..int4),
        authorityEntry                  AccessControlEntry OPTIONAL,
        holderEntry                     AccessControlEntry OPTIONAL
        }
```

The AuthorityAccessControlListEntry type is of type AccessControlListEntry with the holder entry absent.

The number component shall specify the number of the retrieved entry of the access control list of access the transaction is operating on. If this list is empty, its value shall be 0.

When the value of the number component in the transaction argument was less than or equal to the number of entries in the access control list the transaction is operating on, the value of the number component shall be the same as the value of the number component in the argument, i.e. the entry designated in the argument will be retrieved.

In case the value of the number component in the transaction argument was greater than the number of entries in a non-empty access control list the transaction is operating on, the value of the number component shall be equal to the number of entries in the list, i.e. the last entry in the list will be retrieved.

The outOf component shall contain the number of entries of the access control list the transaction is operating on.

The authorityEntry component, if present, specifies the entry in the access control list designated by the value of the number component.

The authorityEntry shall only be present when the entry in the access control list the transaction is operating on is an entry that is added or updated by means of an updateAccessControlList transaction with an authorityUpdate argument.

The holderEntry component, if present, specifies the entry in the access control list designated by the value of the number component.

The holderEntry component shall only be present when the entry in the access control list the transaction is operating on is an entry that is added or updated by means of an updateAccessControlList transaction with a holderUpdate argument.

6.2.16 Get cleartext access control list entry

6.2.16.1 The get cleartext access control list entry transaction

The getCleartextAccessControlListEntry transaction shall be used by or on behalf of the ERT holder to obtain a cleartext entry from the access control list contained in the ERT. The getCleartextAccessControlListEntry transaction is defined as follows:

```
getCleartextAccessControlListEntry TRANSACTION ::= {
        ARGUMENT                    GetCleartextAccessControlListEntryArgument
        RESULT                      CLEAR-SECURED {AccessControlListEntry}
        ERRORS                      {GetCleartextAccessControlListEntryErrors}
        CODE                        10
        }
```

The ARGUMENT field is of type GetCleartextAccessControlListEntryResult.

If the transaction is performed successfully, the ERT returns a CLEAR-SECURED {AccessControlListEntry} value. If not, one of the error codes from GetCleartextAccessControlListEntryErrors is returned.

A CLEAR-SECURED {AccessControlListEntry} value contains the AccessControlListEntry data (see below) signed or not according to the capabilities of the ERT (see 6.2.18.1 for the definition of CLEAR-SECURED {}).

The transaction operates on a list of all access control entries contained in the ERT. The ERT assigns to each entry in this list, if any, a distinct number between 1 and the number of entries in this list.

6.2.16.2 The get cleartext access control list entry transaction argument

The GetCleartextAccessControlListEntryArgument is defined as follows:

```
GetCleartextAccessControlListEntryArgument ::= SEQUENCE {
        credentials                 ErtHolderCredentials,
        challenge                   Challenge,
        number                      INTEGER (1..int4)
        }
```

The credentials component shall contain the credential (VehicleId and PIN) of the ERT holder.

The challenge component is a random number generated by the ERI reader which shall eventually be returned in the signed part of the result to show that the result returned is the result of this invocation and not a replicated message.

The number component specifies the number of the entry to be retrieved from the access control list the transaction is operating on.

6.2.16.3 The get cleartext access control list entry transaction errors

The GetCleartextAccessControlListEntryErrors value set is defined as follows:

```
GetCleartextAccessControlListEntryErrors ErrorCode ::= {
        illegalArgument |           illegalVehicleId |
        illegalHolderAccess |       otherError
        }
```

The GetCleartextAccessControlListEntryError type is of type EriErrorCode and shall take one of the values listed.

6.2.17 Get ERT capabilities

6.2.17.1 The get ERT capabilities transaction

The getErtCapabilities transaction shall be used to obtain the capabilities of the ERT and is defined as follows:

```
getErtCapabilities TRANSACTION ::= {
        ARGUMENT                    NULL
        RESULT                      ErtCapabilities
        CODE                        15
        }
```

The argument field is of type NULL.

The RESULT field is of type ErtCapabilities as defined below.

6.2.17.2 The get ERT capabilities transaction result

The ErtCapabilities type is defined as follows:

```
ErtCapabilities ::= SEQUENCE {
        supportedTransactions           SEQUENCE OF INTEGER,
        hashingAlgorithms               SEQUENCE OF HashingAlgorithm OPTIONAL,
        signingAlgorithms               SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        signatureVerificationAlgorithms SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        secretKeyEncryptionAlgorithms   SEQUENCE OF SecretKeyAlgorithm OPTIONAL,
        publicKeyEncryptionAlgorithms   SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        publicKeyDecryptionAlgorithms   SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        maxBitsPublicKeys               INTEGER (0..int4),
        maxOctetsPin                    INTEGER (0..int4),
        maxOctetsSetArgument            INTEGER (0..int4),
        maxNumberSetArguments           INTEGER (1..int4),
        maxNumberComArguments           INTEGER (1..int4),
        maxSizeAccessControlList        INTEGER (0..int4),
        maxNumberAuthorities            INTEGER (0..int4),
        maxNumberAddServProviders       INTEGER (0..int4),
        ertSecurityIndicationSupport    ErtSecurityFlags,
        maxInteger                      INTEGER (1..int4),
        maxStringSize                   INTEGER (1..int4),
        …
        }
```

NOTE 1 There is no need to signal whether or not a registration authority has disabled encryption and authentication. One can just invoke a get ERI data transaction and look to the result. A registration authority can also determine this by inspecting the argument of the last commissioning transaction by using the get ciphertext historic commissioning data transaction.

The supportedTransactions component specifies the supported transactions identified by the value of their transaction code field.

The hashingAlgorithms component, if present, specifies the supported hashing algorithms.

The hashingAlgorithms component shall be present if the ERT has signing or signature verification capabilities. If not, it shall be absent.

The signingAlgorithms component, if present, specifies the supported public key algorithms for signing.

The signingAlgorithms component shall be present if the ERT has signing capabilities. If not, it shall be absent.

The signatureVerificationAlgorithms component, if present, specifies the supported public key algorithms for signature verification.

The signatureVerificationAlgorithms component shall be present if the ERT has signature verification capabilities. If not, it shall be absent.

The secretKeyEncryptionAlgorithms component, if present, specifies the supported secret key algorithms.

The secretKeyEncryptionAlgorithms component shall be present if the ERT has encryption capabilities. If not, it shall be absent.

The publicKeyEncryptionAlgorithms component, if present, specifies the supported public key algorithms for encryption.

The publicKeyEncryptionAlgorithms component shall be present if the ERT has encryption capabilities. If not, it shall be absent.

The publicKeyDecryptionAlgorithms component, if present, specifies the supported public key algorithms for decryption.

The publicKeyDecryptionAlgorithms component shall be present if the ERT has decryption capabilities. If not, it shall be absent.

The maxBitsPublicKeys component specifies the maximum number of bits that may be used for a public or private key.

If no public key algorithms are supported, the value of the maxBitsPublicKeys component shall be zero.

The maxOctetsPin component specifies the maximum number of characters that may be used for a PIN.

If no PINs are supported the value of the maxOctetsPin component shall be zero.

The maxOctetsSetArgument component specifies the maximum number of octets that may be used for the argument of a set ERI data transaction.

The maxNumberSetArguments component specifies the maximum number of arguments of a set ERI data transaction that can be stored in the ERT.

The maxNumberComArguments component specifies the maximum number of arguments of commissioning transaction that can be stored in the ERT.

The maxSizeAccessControlList component specifies the maximum number of entries of the access control list within the ERT.

The maxNumberAuthorities component specifies the maximum number of entries for authorities in the access control list within the ERT. This number shall be less than or equal to the value of the maxSizeAccessControlList component.

The maxNumberAddServProviders component specifies the maximum number of entries for additional service providers in the access control list within the ERT. This number shall be less than or equal to the value of the maxSizeAccessControlList component.

The sum of the values of the maxNumberAuthorities and the maxNumberAddServProviders components may be more than the value of the maxSizeAccessControlList component.

The ertSecurityIndicationSupport component specifies the support of the ERT security capabilities and is of type ErtSecurityFlags. A “1” bit indicates that the corresponding capability is supported, a “0” bit that it is not supported.

NOTE 2 The same type is used for the ertSecurityStatus component in the CleartextEriData type but with a different meaning attached to the bits.

The maxInteger component specifies the maximum value of an integer that can be handled by the ERT.

The maxStringSize component specifies the maximum number of octets that may be used for string in this ERT.

6.2.18 Common definitions

6.2.18.1 ERT tagged and secured data

6.2.18.1.1 SECURED data

The parameterized type SECURED shall be used by an ERT to secure data according to its capabilities and is defined as follows:

```
SECURED {ToBeSecured} ::= CHOICE {
        authenticatedAndEncrypted    ENCRYPTED {ERT-AUTHENTICATED {TAGGED {ToBeSecured}}},
        authenticated                ERT-AUTHENTICATED {TAGGED {ToBeSecured}},
        encrypted                    ENCRYPTED {TAGGED {ToBeSecured}},
        cleartext                    TAGGED {ToBeSecured}
        }
```

For all alternatives, the parameterized type TAGGED adds the unique number of the ERT and, if applicable, the challenge and a sequence number. (See the definition of the parameterized type TAGGED below.)

The authenticatedAndEncrypted alternative shall be used if the ERT is commissioned and has both enabled encipherment and enabled signing capabilities.

The authenticatedAndEncrypted alternative, if present, shall contain the tagged data TAGGED{ToBeSecured} authenticated and subsequently encrypted by the ERT.

The authenticated alternative shall be used if the ERT is commissioned and has enabled signing capabilities, but does not have enabled encipherment capabilities.

The authenticated alternative, if present, shall contain the tagged data TAGGED{ToBeSecured} authenticated by the ERT.

The encrypted alternative shall be used if the ERT is commissioned and has enabled encipherment capabilities, but does not have enabled signing capabilities.

The encrypted alternative, if present, shall contain the tagged data TAGGED{ToBeSecured} encrypted by the ERT.

The cleartext alternative shall be used if the ERT is not commissioned or is commissioned and has neither enabled encipherment capabilities nor enabled signing capabilities.

The cleartext alternative, if present, shall contain the tagged data TAGGED{ToBeSecured}.

The public encipherment key used for encryption shall be that of the entity identified in the onBehalfOf component in the argument of the invoked transaction or, if not available, the public encipherment key of the registration authority that has commissioned the ERT.

The private signature key used for authentication shall be the one provided in the last commissioning transaction.

6.2.18.1.2 CLEAR SECURED data

The parameterized type CLEAR-SECURED shall be used by an ERT to secure cleartext data according to its capabilities and is defined as follows:

```
CLEAR-SECURED {ToBeSecured} ::= SECURED {ToBeSecured} (WITH COMPONENTS
        {authenticatedAndEncrypted ABSENT, encrypted ABSENT} )
```

The parameterized type CLEAR-SECURED can take the same values as the parameterized type SECURE but with the alternatives authenticatedAndEncrypted and encrypted not being used.

6.2.18.1.3 Tagged data

The parameterized type TAGGED shall be used to add the ERT number and transaction specific data to the result of a read transaction before this result is signed, if applicable, or encrypted, if applicable. The parameterized type TAGGED is defined as follows:

```
TAGGED {ToBeTagged} ::= SEQUENCE {
        ertNumber                   ErtNumber,
        challenge                   Challenge OPTIONAL,
        sequenceNumber              INTEGER (1..int4) OPTIONAL,
        tobeTagged                  ToBeTagged
        }
```

The ertNumber component shall contain the unique number of the ERT.

The challenge component, if present, shall contain the value of the challenge parameter from the argument of the invoked transaction.

The challenge component shall be present when both conditions are fulfilled: the ERT has enabled signing capabilities and the challenge was present in the argument of the transaction invocation.

The challenge component may be present when the ERT has no enabled signing capabilities but the challenge was present in the argument of the transaction invocation.

The challenge component shall not be present when not present in the argument of the transaction invocation.

NOTE 1 The challenge component guarantees that the signed data is not a replicated message but indeed the result of this particular invocation of the transaction.

The sequenceNumber component, if present, shall contain the sequence number of the read transaction. This number shall start with 1 and shall be increased by 1 after being used in the result of a transaction.

If the sequence number reaches its maximum value, its next value shall be 1.

The sequenceNumber component shall be present when the ERT has enabled encryption capabilities.

The sequenceNumber component may or may not be present when the ERT has no enabled encryption capabilities.

NOTE 2 This sequence number guarantees that no two ERT encrypted messages are the same, not even if the same challenge is used. This prevents, when encrypted, the occurrence of “pseudonyms”, i.e. messages which unintentionally identify a vehicle only because the messages are the same.

6.2.18.2 ERT authentication definitions

6.2.18.2.1 ERT authentication

The parameterized type ERT-AUTHENTICATED is defined as follows:

```
ERT-AUTHENTICATED {ToBeErtAuthenticated} ::= SEQUENCE {
        ertSigned                       SIGNED {ToBeErtAuthenticated, PrivateSignatureKey},
        publicVerificationKeyCertificate ErtCertificate
        }
```

The ertSigned component contains the authenticated data, i.e. the data signed with the private signature key of the ERT.

The publicVerificationKeyCertificate component shall contain a public key certificate issued by the vehicle's registration authority for the public verification key to be used to check ERT signatures.

NOTE This public key certificate should be written into the ERT as part of a commission ERT transaction.

6.2.18.2.2 The ERT certificate

The ErtCertificate type is defined as follows:

```
ErtCertificate ::= SIGNED {ErtCertificationData, PrivateSignatureKey}
```

The ErtCertificationData shall contain the contents of the ERT certificate.

The ErtCertificationData shall be signed with the private signature key of the registration authority.

NOTE 1 It is not required that the hashing algorithm and public key algorithm be supported by the ERT. An ERT certificate is provided by the registration authority in a commission ERT transaction and returned by the ERT in read transactions. An ERT certificate is neither policed nor interpreted by an ERT.

NOTE 2 Nevertheless, as an ERI reader should be able to use these certificates issued by the registration authority, both the hashing algorithm and public key algorithm should be supported by this part of ISO 24534.

6.2.18.2.3 The ERT certificate data

The type ErtCertificationData shall be used for the data to be certified and is defined as follows:

```
ErtCertificationData ::= SEQUENCE {
        vehicleId               VehicleId,
        publicVerificationKey   PublicVerificationKey,
        signatoryId             EntityId,
        date                    DATE
        }
```

The vehicleId component contains the vehicleId.

The publicVerificationKey component contains the public verification key for the verification of ERT signatures.

The signatoryId contains the ID of the registration authority that has issued the certificate.

The date component contains the date at which the certificate was signed.

6.2.18.3 BOE authentication definitions

6.2.18.3.1 BOE dated

The parameterized type DATED shall be used to attach validity information to data and is defined as follows:

```
DATED {ToBeDated} ::= SEQUENCE {
        date                    DATE,
        validThru               DATE OPTIONAL,
        issuer                  EntityId,
        toBeDated               ToBeDated
        }
```

The data component shall contain the date at which the value of the ToBeDated parameter is signed.

The value of the ToBeDated parameter becomes valid at the time contained in the date component.

The validThru component, if present, specifies the date until which the signature for the value of the ToBeDated parameter is valid.

When the validThru component is absent, the value of the ToBeDated parameter is valid for an unlimited period of time.

The issuer component identifies the issuer of the value of the ToBeDated parameter, i.e. the manufacturer or registration authority that has signed this value.

The toBeDated component contains the value of the ToBeDated parameter.

6.2.18.3.2 BOE authentication

The parameterized type BOE-AUTHENTICATED shall be used for the authentication of data issued by a registration authority and is defined as follows:

```
BOE-AUTHENTICATED {ToBeBoeAuthenticated} ::= SEQUENCE {
        signedParameter                 SIGNED {ToBeBoeAuthenticated, PrivateSignatureKey},
        certificates                    SEQUENCE SIZE(1..2) OF BoeCertificate
        }
```

The signedParameter component shall contain the value of the ToBeBoeAuthenticated parameter signed with the private signature key of the registration authority.

The certificates component shall contain a sequence of one or two public key certificates of type BoeCertificate as defined below.

When the certificates component contains only one public key certificate, this certificate must be issued by the top-level certification authority for the public signature key of the registration authority whose data has to be authenticated.

When the certificates component contains two public key certificates, the first one is issued by the top-level certification authority for the public signature key of the intermediate certification authority which signed the second certificate. The second certificate is issued by an intermediate certification authority for the public signature key of the registration authority whose data has to be authenticated.

The public key certificates in the sequence shall be valid at the date at which the value of the ToBeBoeAuthenticated parameter is signed.

6.2.18.3.3 The BOE certificate

The BoeCertificate type is defined as follows:

```
BoeCertificate ::= SIGNED {BoeCertificationData, PrivateSignatureKey}
```

The BoeCertificationData shall contain the BoeCertificationData.

The BoeCertificationData shall be signed by a certification authority using a hashing algorithm and a public key algorithm supported by the ERT.

NOTE 1 The ERT has to verify a registration authority signature using one or more BOE certificates.

NOTE 2 The list of supported algorithms can be obtained by invoking a get ERT capabilities transaction.

6.2.18.3.4 BOE certification data

The CertificationData type is defined as follows:

```
BoeCertificationData ::= SEQUENCE {
        entityId                    EntityId,
        entityRole                  EntityRole,
        entityName                  Text OPTIONAL,
        publicKey                   Key,
        signatoryId                 EntityId,
        signatoryName               Text OPTIONAL,
        signatoryRole               EntityRole,
        date                        DATE,
        validThru                   DATE
        }
```

The entitityId component identifies the certified entity.

The entityRole component specifies the role of the certified entity.

The entityName component, if present, contains the name of the certified entity in a language chosen by the signatory of the certificate.

The publicKey component contains a public key of the certified entity.

The signatoryId component identifies the signatory of the certificate.

The signatoryName component, if present, shall contain the name of the signatory in the language of their choice.

The signatoryRole component specifies the role of the signatory.

The date component contains the date at which the certificate was signed.

The validThru component specifies the date until which the certificate is valid.

6.2.18.3.5 The entity roles

The EntityRole type specifies the role of an entity and is defined as follows:

```
EntityRole::= ENUMERATED {
        topLevelCertificationAuthority (0), intermediateCertificationAuthority (1),
        manufacturer (2),                   registrationAuthority (3),
        authority (4),                      serviceProvider (5),
        eriHolder (6)                       }
```

The EntityRole type can take one of the defined values.

6.2.18.4 Signing and signatures

6.2.18.4.1 Signed

The parameterized type SIGNED shall be used to sign data and is defined as follows:

```
SIGNED {ToBeSigned, PrivateSignatureKey} ::= SEQUENCE {
        toBeSigned                  ToBeSigned,
        hashingAlgorithm            HashingAlgorithm DEFAULT sha1,
        signatureAlgorithm          PublicKeyAlgorithm DEFAULT ellipticCurve,
        signature                   SIGNATURE {ToBeSigned, HashingAlgorithm,
                                    PublicKeyAlgorithm, PrivateSignatureKey}
        }
```

The toBeSigned component shall contain the value of the SIGNED parameter ToBeSigned.

The PrivateSignatureKey parameter shall contain the value of the private signature key used to sign the toBeSigned component.

The hashingAlgorithm component, if present, shall specify the hashing algorithm used to calculate a hash-code over the ToBeSigned parameter.

If not present, the default value is sha1.

The signatureAlgorithm component, if present, shall specify the public key algorithm used to sign the hash-code over the ToBeSigned parameter.

If not present, the default value is ellipticCurve.

The signature component shall contain the signature for the value of the ToBeSigned parameter using the hashingAlgorithm and signatureAlgorithm (see 6.2.18.4.2).

6.2.18.4.2 The signature

The parameterized type SIGNATURE is defined as follows:

```
SIGNATURE {ToBeSigned, HashingAlgorithm, SignatureAlgorithm, PrivateSignatureKey} ::= BIT STRING
        (CONSTRAINED BY
{HashingAlgorithm, -- and -- SignatureAlgorithm, -- with -- PrivateSignatureKey, -- applied to -- ToBeSigned} )
```

The value of SIGNATURE is the BIT STRING which results from a two-step process. First, a hash-code is calculated with the hashing algorithm over the encoded value of the ToBeSigned parameter. Then this hash-code is encrypted with the public key signature algorithm using the value of the PrivateSignatureKey parameter as the private signature key.

6.2.18.4.3 The challenge parameter

The challenge type is defined as follows:

```
Challenge ::= INTEGER (1..int4)
```

6.2.18.5 Common encryption definitions

6.2.18.5.1 ERT encryption

The parameterized type ENCRYPTED is defined as follows:

```
ENCRYPTED {ToBeErtEncrypted} ::= SEQUENCE {
        enciphermentKeyId               KeyId,
        publicKeyEncryptionAlgorithm    PublicKeyAlgorithm DEFAULT ellipticCurve,
        secretKeyEncryptionAlgorithm    SecretKeyAlgorithm DEFAULT aes,
        encryptedKey                    KEY-ENCRYPTION {SecretTransactionKey,
                                        PublicKeyAlgorithm, PublicEnciphermentKey},
        ciphertext                      CIPHER-TEXT {ToBeErtEncrypted,
                                        SecretKeyAlgorithm, SecretTransactionKey}
        }
```

The enciphermentKeyId component identifies the public encipherment key used for the encryption of the secret transaction key. Its value shall be used to determine the corresponding private decipherment key.

If the encryption is performed by an ERT with enabled encipherment capabilities and no public encipherment key is available, the keyId shall take a random value.

NOTE 1 In normal circumstances an ERT should never have to use a random value for the keyId. However, if no key is available, a random value makes it more difficult for an illegal reader to notice that no key was available.

The publicKeyEncryptionAlgorithm component, if present, shall identify the public key algorithm used for the encryption of the randomly selected secret key used to encrypt the data ToBeEncrypted. If not present, the default value is the elliptic curve algorithm.

The secretKeyEncryptionAlgorithm component, if present, shall identify the secret key algorithm used to encrypt the data ToBeEncrypted. If not present, the default value is the AES algorithm.

The encryptedKey component contains a secret random transaction key encrypted with a public encipherment key.

NOTE 2 In time-critical situations an ERT may encrypt this secret transaction key before the transaction for which it is needed is invoked.

The ciphertext component contains the result of encrypting the ToBeEncrypted parameter with the secret transaction key.

NOTE 3 For performance reasons, ERI data is encrypted with a once-only symmetric key which is encrypted with the asymmetric public encipherment key.

6.2.18.5.2 Public key encryption

The parameterized type KEY-ENCRYPTION is defined as follows:

```
KEY-ENCRYPTION {KeyToBeEncrypted, PublicKeyAlgorithm, PublicEnciphermentKey} ::=
        BIT STRING ( CONSTRAINED BY {
        PublicKeyAlgorithm, -- with -- PublicEnciphermentKey, -- applied to -- KeyToBeEncrypted
        -- or random bit string with same length if no public key is available --} )
```

The value of the BIT STRING is the result of a public key encryption of the value of the ToBeEncrypted parameter using the public key algorithm identified by the PublicKeyAlgorithm parameter and the value of the PublicEnciphermentKey parameter as the public encipherment key.

If the encryption is performed by an ERT with enabled encipherment capabilities and no public encipherment key is available, the BIT STRING shall take a random value of the same length.

6.2.18.5.3 Secret key encryption

The parameterized type CIPHER-TEXT is defined as follows:

```
CIPHER-TEXT {ToBeEncrypted, SecretKeyAlgorithm, SecretKey} ::=
        BIT STRING ( CONSTRAINED BY {
        SecretKeyAlgorithm, -- with -- SecretKey, -- applied to -- ToBeEncrypted
        -- or random bit string with same length if no public key is available --} )
```

The value of the BIT STRING is the result of encrypting the value of the ToBeEncrypted parameter using the secret key algorithm identified by the SecretKeyAlgorithm parameter and the value of the SecretKey parameter as the (symmetric) secret encipherment key.

If the encryption is performed by an ERT with enabled encipherment capabilities and no public encipherment key is available, the BIT STRING shall take a random value of the same length.

6.2.18.6 Other common definitions

6.2.18.6.1 ERT holder credentials

The type ErtHolderCredentials is defined as follows:

```
ErtHolderCredentials ::= SEQUENCE {
        vehicleId                       VehicleId,
        pin                             PIN
        }
```

The component vehicleId contains the VehicleId of the vehicle.

The component pin contains the PIN for the ERT holder.

6.2.18.6.2 Cryptographic algorithms

6.2.18.6.2.1 Hashing algorithms

The set of supported hashing algorithms is defined as follows:

```
HashingAlgorithm ::= ENUMERATED {
        sha1,
        ... }
```

Currently only the Secure Hashing Algorithm 1 (see FIPS 180-2 and ISO/IEC 10118-3) is supported.

NOTE The definition is only supplied to support future extensions of this part of ISO 24534.

6.2.18.6.2.2 Public key algorithms

The set of supported public key algorithms is defined as follows:

```
PublicKeyAlgorithm  ::= ENUMERATED {
        ellipticCurve,
        ... }
```

Currently only the elliptic curve algorithm (see ISO/IEC 15946 and ISO/IEC 18033-2) is supported.

NOTE The definition is only supplied to support future extensions of this part of ISO 24534.

6.2.18.6.2.3 Secret key algorithms

The set of supported secret key algorithms is defined as follows:

```
SecretKeyAlgorithm  ::= ENUMERATED {
        aes,
        ... }
```

Currently only the AES algorithm (see FIPS 197 and ISO/IEC 18033-3) is supported.

NOTE The definition is only supplied to support future extensions of this part of ISO 24534.

6.2.18.6.3 Keys and key identifiers

The various key types are defined as follows:

```
SecretTransactionKey            ::= Key
PublicEnciphermentKey           ::= Key
PrivateDeciphermentKey          ::= Key
PrivateSignatureKey             ::= Key
PublicVerificationKey           ::= Key
Key                             ::= BIT STRING

PIN                             ::= NumericString (SIZE(4))

KeyId                           ::= INTEGER (0..65535)
```

The type KeyId is of type integer and shall be used to identify a key. It can take any value between 0 and 65535.

6.2.18.6.4 ERT number

The ErtNumber type is defined as follows:

```
ErtNumber ::= INTEGER
```

6.2.18.6.5 Text

The type Text shall be used for a textual description in a language chosen by the user and is defined as follows:

```
Text ::= UTF8String (SIZE (1..256))
```

6.2.18.7 Int4

The value int4 is defined as follows:

```
int4 INTEGER ::= 4294967295
```

6.2.18.8 ERI error codes

The EriErrorCodes type is defined as follows:

```
ErrorCode ::= ENUMERATED {
        illegalArgument,                illegalVehicleId,
        illegalCertificate,             illegalSignature,
        illegalEntity,                  illegalHolderAccess,
        illegalDate,                    notCustomized,
        notCommissioned,                noEntry,
        resourceLimitExceeded,
-- encipherment support errors
        noEnciphermentCapability,
        noDeciphermentCapability,
        secretKeyEncryptionAlgorithmNotSupported,
        publicKeyEncryptionAlgorithmNotSupported,
-- authentication support errors
        noSigningCapability,
        noSignatureVerificationCapability,
        hashingAlgorithmNotSupported,
        signingAlgorithmNotSupported,

        otherError,
        … }
```

The EriErrorCode type can take one of the following values:

— illegalArgument: if the argument has an illegal structure or an illegal value;

— illegalVehicleId: if the VIN, if present, is not conformant to ISO 3779 or the vehicleId does not contain the required value;

— illegalCertificate: if one or more certificates are illegal or do not contain correct values;

— illegalSignature: if the signature is not correct;

— illegalEntity: if the entityId or entity role is incorrect;

— illegalHolderAccess: if the credentials of the ERT holder were not correct;

— illegalDate: if the date was not correct, e.g. not within the validity period of the certificate(s) or not later than in earlier authentication data;

— notCustomized: if the ERT is not yet customized (and does not yet contain a vehicle identifier);

— notCommissioned: if the ERT is customized but not commissioned;

— noEntry: if the entry was not present, when there is e.g. no entry with a particular entry ID;

— resourceLimitExceeded: if an ERT cannot store the data in the argument of a transaction because the size of the data exceeds its storage capacity;

— noEnciphermentCapability: if a transaction requires encipherment capabilities and the ERT has no enabled encipherment capabilities;

— noDeciphermentCapability: if a transaction requires decipherment capabilities and the ERT has no decipherment capabilities;

— secretKeyEncryptionAlgorithmNotSupported: if a transaction specifies a secret key encryption algorithm that is not supported by the ERT;

— publicKeyEncryptionAlgorithmNotSupported: if a transaction specifies a public key algorithm for encryption or decryption that is not supported by the ERT;

— noSigningCapability: if a transaction requires signing capabilities while the ERT has no enabled signing capabilities;

— noSignatureVerificationCapability: if a transaction requires signature verification capabilities while the ERT has no signature verification capabilities;

— hashingAlgorithmNotSupported: if a transaction specifies a hashing algorithm that is not supported by the ERT;

— signingAlgorithmNotSupported: if a transaction specifies a public key algorithm for signing or signature verification that is not supported by the ERT;

— OtherError: if some other error has occurred while the transaction was attempted.

6.3 The ERT interfaces

6.3.1 General ERT interface conformance requirements

The ERI data, ERI security data and ERT number can only be accessed as specified in this part of ISO 24534.

An application layer protocol data unit to be exchanged with an ERT shall be an ERI protocol data unit of type EriPdu, i.e. of type EriRequestPdu or of type EriResponsePdu.

An ERI protocol data unit shall be encoded in conformance with the canonical Packed Encoding Rules (CANONICAL-PER) ALIGNED variant defined in ISO/IEC 8825-2.

The lower layer protocols (session and lower, as applicable) shall comply with International Standards.

NOTE If required, an ERI protocol data unit may be segmented and reassembled (in ISO/IEC 7498-1 terminology) as appropriate.

6.3.2 The ISO/IEC 14443 interface

In the case where the interface between an ERT and an onboard or external proximity ERI reader/writer is based on ISO/IEC 14443, 6.3.1 applies and the interface shall comply with ISO/IEC 14443 parts 1, 2, 3, and 4, and with:

— the ERT acting as a PICC (proximity integrated circuit card) of type A or B;

— the onboard ERI reader/writer acting as a PCD (proximity coupling device) supporting both types A and B.

An ERI protocol data unit shall be directly transferred using the INF field of one or more I-blocks (see ISO/IEC 14443-4).

NOTE 1 Consequently, an ERI protocol data unit is not packed into ISO/IEC 7816-4 application protocol data units as suggested in ISO/IEC 14443-4.

Segmenting and reassembly of an ERI protocol data unit shall be accomplished, if required, with chaining as specified in ISO/IEC 14443-4.

Collisions between an onboard reader or writer and an external (handheld) reader or writer shall be avoided.

NOTE 2 For a handheld reader or writer this may be accomplished if the other onboard ERI equipment is switched off when the engine is switched off and/or when the vehicle is not being driven.

6.3.3 The short-range air interface

6.3.3.1 General short-range air interface requirements

A short-range air interface shall be capable of exchanging ERI protocol data units conforming to 6.3.1.

An onboard communication device providing short-range access to an ERT shall be capable of transferring ERI protocol data units received from its (cellular network) peer to the ERT and vice versa.

6.3.3.2 The use of the DSRC application layer protocol

6.3.3.2.1 General

If the DSRC application layer protocol is used for ERI transactions, ISO 15628 shall be applied as specified in this clause.

NOTE This makes the ERI DSRC interface compatible with other DSRC application interfaces such as ISO 14906.

6.3.3.2.2 Use of the DSRC initialization service

Whenever a DSRC link is to be used for ERI transactions, the ISO 15628 initialization service shall be used as follows:

— Either the mandApplications component or the nonmandApplications component of the initialization-request T-PDU (Beacon Service Table, BST) shall contain an ERI application component.

— The applications component of the initialization-response T-PDU (Vehicle Service Table, VST) shall contain an ERI application component.

— The value of the ERI application component in an initialization-request or an initialization-response shall be as follows:

  — The aid component shall have the value “automatic-vehicle-identification”.

  — The eid component may be omitted and, if present, shall be ignored by the ERI application.

  — The parameter component may be omitted and, if present, shall be ignored by the secure communications functions using asymmetric techniques as defined in this part of ISO 24534.

NOTE 1 The designation of an application as mandatory or non-mandatory and its position in the list of applications are outside the scope of this part of ISO 24534. They only influence the priority of the ERI application relative to other applications identified in the BST (see 7.3.2.2 of ISO 15628:2007).

NOTE 2 The eid component and the parameter component may however be used for other, non-ERI, AVI applications.

6.3.3.2.3 Use of the DSRC action-request

An ERI transaction request is sent from an ERI reader/writer to the onboard DSRC unit as an ISO 15628 action-request as follows:

— The value of the mode component shall be TRUE (as all ERI transactions are confirmed).

— The value of the eid component shall be 0.

— The value of the actionType component shall be eriTransaction.

— The accessCredentials component shall not be present.

— The value of the accessParameter component shall be passed as received to the ERT as the value of an eriRequestPdu.

— The iid component shall not be present.

NOTE The action-request is of type Action-Request which is defined in ISO 15628 as follows:

```
Action-Request::=SEQUENCE{
        mode                    BOOLEAN,
        eid                     Dsrc-EID,
        action                  Type ActionType,
        accessCredentials       OCTET STRING (SIZE (0..127,...)) OPTIONAL,
        actionParameter         Container OPTIONAL,
        iid                     Dsrc-EID OPTIONAL
        }
```

(end of note)

6.3.3.2.4 Use of the DSRC action response

An ERI transaction response received from an ERT is sent by the onboard DSRC unit to the external ERI reader as an ISO 15628 action-response as follows:

— The value of the eid component shall be 0.

— The iid component shall not be present.

— The value of the responseParameter component, if present, shall be the value of the eriResponsePdu as received from the ERT.

— The ret component may be omitted and, if present, shall be ignored when the eriResponsePdu is also present.

NOTE 1 The action-response shall be of type Action-Response which is defined in ISO 15628 as follows:

```
Action-Response::=SEQUENCE{
        Fill                    BIT STRING (SIZE(1)),
        eid                     Dsrc-EID,
        iid                     Dsrc-EID OPTIONAL,
        responseParameter       Container OPTIONAL,
        ret                     ReturnStatus OPTIONAL
        }
```

(end of note)

In case the DSRC device is not capable of transferring an eriRequestPdu to an ERT, an ISO 15628 action-response containing a ret component of type ReturnStatus is returned to the roadside unit.

NOTE 2 The mechanisms to be used for passing an EriRequestPdu from a DSRC device to the ERI device are outside the scope of this part of ISO 24534. It is assumed that some generic onboard platform or network will emerge that can be used for this purpose. In the meantime, the manufacturer of a DSRC device may have to cope with different means for connecting its DSRC device to the onboard reader/writer of the ERI unit.

6.3.3.2.5 Embedding of ERI provisions in the DSRC application layer definition

See the ASN.1 module “Reduced ISO 15628 MODULE” in A.2.

6.3.4 The remote access interface

A short-range air interface shall be capable of exchanging ERI data protocol units in conformance with 6.3.1.

An onboard communication device providing remote access to an ERT shall be capable of transferring ERI protocol data units received from its (cellular network) peer to the ERT and vice versa.

Annex A
(normative)

ASN.1 modules

A.1 Overview

This annex contains the following ASN.1 modules:

— Transaction module (starts at page 66);

— A reduced ISO 15628 module to show how it can be used (starts at page 76).

The vehicle data module "ElectronicRegistrationIdentificationVehicleDataModule" can be found in ISO 24534-3.

A.2 Modules

NOTE This clause can as a whole be converted to simple text and then be compiled. It therefore contains no additional clause headers and titles.

-- TRANSACTIONS MODULE –

```
ElectronicRegistrationIdentificationTransactionsModule
 {iso(1) standard(0) iso24534 (24534) transactions (2) version (0)}
DEFINITIONS AUTOMATIC TAGS ::= BEGIN

-- Electronic Registration Identification (ERI) Transactions

-- EXPORTS  everything;

IMPORTS
        RegistrationAuthority, VehicleId, AdditionalEriData, EntityId
                                        FROM ElectronicRegistrationIdentificationVehicleDataModule;

EriPdu ::= CHOICE {
        requestPdu                      EriRequestPdu,
        reponsePdu                      EriResponsePdu
        }

EriRequestPdu ::= SEQUENCE {
        transactCode                    TRANSACTION.&transactionCode ( {EriTransactions}),
        argument                        TRANSACTION.&ArgumentType
                                        ( {EriTransactions} {@.transactCode}) OPTIONAL

        }

EriResponsePdu ::= CHOICE {
        resultPdu                       EriResultPdu,
        errorPdu                        EriErrorPdu
        }
```

```
EriResultPdu ::= SEQUENCE {
        transactCode                TRANSACTION.&transactionCode ( {EriTransactions}),
        result                      TRANSACTION.&ResultType ( {EriTransactions} {@.transactCode})
        }

EriErrorPdu ::= SEQUENCE {
        transactCode                TRANSACTION.&transactionCode ( {EriTransactions}),
        error                       TRANSACTION.&ErrorCodes ( {EriTransactions} {@.transactCode})
        }
```

-- TRANSACTIONS

```
TRANSACTION ::= CLASS {
        &ArgumentType               ,
        &ResultType                 ,
        &ErrorCodes                 ErrorCode OPTIONAL,
        &transactionCode            INTEGER UNIQUE
        }
WITH SYNTAX {
        ARGUMENT                    &ArgumentType
        RESULT                      &ResultType
        [ERRORS                     &ErrorCodes]
        CODE                        &transactionCode
        }

EriTransactions TRANSACTION ::= {
        getEriData | getAuthenticatedEriData |
        setEriData | getCiphertextHistoricEriData | getCleartextHistoricEriData |
        getPublicCertificateVerificationKeyId | getPublicEnciphermentKeyErt |
        commissionErt | withdrawCommissioning |
        getCiphertextHistoricComData | getCleartextHistoricComData |
        updateAccessControlList | getCiphertextAccessControlListEntry | getCleartextAccessControlListEntry |
        getErtCapabilities
        }
```

-- Get ERI data

```
getEriData TRANSACTION ::= {
        ARGUMENT                    GetEriDataArgument
        RESULT                      GetEriDataResult
        ERRORS                      {notCustomized}
        CODE                        1
        }

GetEriDataArgument ::= SEQUENCE {
        onBehalfOf                  EntityId OPTIONAL,
        challenge                   Challenge,
        includeAdditionalData       BOOLEAN
        }

GetEriDataResult ::= SEQUENCE {
        registrationAuthority       RegistrationAuthority OPTIONAL,
        eriResultData               SECURED {CleartextEriData}
        }
```

```
-- Authenticate ERI data

getAuthenticatedEriData TRANSACTION  ::= {
        ARGUMENT                    GetAuthenticatedEriDataArgument
        RESULT                      GetAuthenticatedEriDataResult
        ERRORS                      {notCustomized}
        CODE                        2
        }

GetAuthenticatedEriDataArgument ::= SEQUENCE {
        ertHolderCredentials        ErtHolderCredentials,
        challenge                   Challenge,
        includeAdditionalData       BOOLEAN
        }

GetAuthenticatedEriDataResult ::= SEQUENCE {
        registrationAuthority       EntityId OPTIONAL,
        authenticateResultData      CLEAR-SECURED {CleartextEriData}
        }

-- ERI data and ERT security flags

CleartextEriData ::= SEQUENCE {
        eriDataOrId                 EriDataOrId,
        ertSecurityStatus           ErtSecurityFlags OPTIONAL
        }

EriDataOrId ::= CHOICE {
        vehicleId                   VehicleId,
        unsignedDatedEriData        DATED {EriData},
        datedAndSignedEriData       SIGNED {DATED {EriData}, PrivateSignatureKey} -- BOE signed
        }

EriData ::= SEQUENCE {
        id                          VehicleId,
        additionalEriData           OCTET STRING (CONTAINING AdditionalEriData) OPTIONAL
        }

ErtSecurityFlags ::= BIT STRING {
        flagsHaveBeenResetted (0),
        notCommissioned (1),
        lowSupplyVoltageIndication (2),
        highSupplyVoltageIndication (3),
        lowClockFrequencyIndication (4),
        highClockFrequencyIndication (5),
        lowTemperatureIndication (6),
        highTemperatureIndication (7)
        } (SIZE (0..16))            -- bit 8 .. 15 reserved for future use

-- Set ERI data

setEriData TRANSACTION ::= {
        ARGUMENT                    SetEriDataArgument
        RESULT                      NULL
        ERRORS                      {SetEriDataErrors}
        CODE                        3
        }

SetEriDataArgument ::= CHOICE {
        clearTextArgument           ClearTextSetEriDataArgument,
        encryptedArgument           ENCRYPTED {ClearTextSetEriDataArgument}
        }
```

```
ClearTextSetEriDataArgument ::= CHOICE {
        authenticatedEriData            BOE-AUTHENTICATED {DATED {EriData}},
        notAuthenticatedEriData         DATED {EriData}
        }

SetEriDataErrors ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalCertificate |            illegalSignature |
        illegalDate |                   notCommissioned |
        resourceLimitExceeded |         otherError
        }
```

-- Retrieve historic ERI data

```
getCiphertextHistoricEriData TRANSACTION ::= {
        ARGUMENT                        GetCiphertextHistoricEriDataArgument
        RESULT                          SECURED {HistoricEriData}
        ERRORS                          {notCustomized}
        CODE                            4
        }

GetCiphertextHistoricEriDataArgument ::= SEQUENCE {
        onBehalfOf                      EntityId OPTIONAL,
        challenge                       Challenge,
        number                          INTEGER (1..int4)
        }

getCleartextHistoricEriData TRANSACTION ::= {
        ARGUMENT                        GetCleartextHistoricEriDataArgument
        RESULT                          CLEAR-SECURED {HistoricEriData}
        ERRORS                          {notCustomized}
        CODE                            5
        }

GetCleartextHistoricEriDataArgument ::= SEQUENCE {
        credentials                     ErtHolderCredentials,
        challenge                       Challenge,
        number                          INTEGER (1..int4)
        }

HistoricEriData ::= SEQUENCE {
        number                          INTEGER (1..int4),
        outOf                           INTEGER (1..int4),
        historicRecord                  ClearTextSetEriDataArgument
        }
```

-- Get public verification key

```
getPublicCertificateVerificationKeyId TRANSACTION ::= {
        ARGUMENT                        NULL
        RESULT                          KeyId
        CODE                            6
        }

getPublicEnciphermentKeyErt TRANSACTION ::= {
        ARGUMENT                        BOE-AUTHENTICATED {VehicleId}
        RESULT                          PublicEnciphermentKey
        ERRORS                          {GetPublicEnciphermentKeyErrors}
        CODE                            7
        }
```

```
GetPublicEnciphermentKeyErrors  ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalCertificate |            illegalSignature |
        illegalEntity |                 noDeciphermentCapability |
        otherError                      }
```

-- Commissioning

```
commissionErt TRANSACTION ::= {
        ARGUMENT                        CommissionErtArgument
        RESULT                          NULL
        ERRORS                          {CommissionErtErrors}
        CODE                            8
        }

CommissionErtArgument ::= CHOICE {
        authenticatedData               BOE-AUTHENTICATED {DATED {CommissioningData}},
        notAuthenticatedData            DATED {CommissioningData}
        }

CommissioningData ::= SEQUENCE {
        vehicleId                       VehicleId,
        registrationAuthority           EntityId,
        resetSecurityFlags              BOOLEAN,
        enciphermentKeyId               KeyId OPTIONAL,
        publicEnciphermentKeyAuthority  PublicEnciphermentKey OPTIONAL,
        publicVerificationKeyCertificate ErtCertificate OPTIONAL,
        privateData                     ENCRYPTED {PrivateCommissioningData} OPTIONAL
        }

PrivateCommissioningData ::= SEQUENCE {
        privateSignatureKeyErt          PrivateSignatureKey OPTIONAL,
        pin                             PIN OPTIONAL
        }

CommissionErtErrors ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalCertificate |            illegalSignature |
        illegalEntity |                 illegalDate |
        notCustomized |                 resourceLimitExceeded |
        noEnciphermentCapability |
        secretKeyEncryptionAlgorithmNotSupported |
        publicKeyEncryptionAlgorithmNotSupported |
        noSigningCapability |
        hashingAlgorithmNotSupported |
        signingAlgorithmNotSupported |
        otherError                      }
```

-- Withdraw Commissioning

```
withdrawCommissioning TRANSACTION ::= {
        -- this transaction also removes all public encipherment keys
        ARGUMENT                        WithdrawCommissioningArgument
        RESULT                          SECURED {WithdrawCommissioningResultData}
        ERRORS                          {WithdrawCommissioningErrors}
        CODE                            withdrawCommissioningCode
        }

withdrawCommissioningCode INTEGER ::= 9
```

```
WithdrawCommissioningArgument ::= CHOICE {
        authenticatedData           BOE-AUTHENTICATED {VehicleId},
        notAuthenticatedData        VehicleId
        }

WithdrawCommissioningResultData ::= [APPLICATION withdrawCommissioningCode ] SEQUENCE {
        withdrawn                   WithdrawCommissioningArgument,
        historicComData             HistoricComData
        }

WithdrawCommissioningErrors ErrorCode ::= {
        illegalArgument |           illegalVehicleId |
        illegalCertificate |        illegalSignature |
        illegalEntity |             illegalDate |
        notCustomized |             notCommissioned |
        otherError                  }
```

-- Get historic commissioning data

```
getCiphertextHistoricComData TRANSACTION ::= {
        ARGUMENT                    GetCiphertextHistoricComDataArgument
        RESULT                      SECURED {HistoricComData}
        ERRORS                      {notCommissioned}
        CODE                        9
        }

GetCiphertextHistoricComDataArgument ::= SEQUENCE {
        onBehalfOf                  EntityId OPTIONAL,
        challenge                   Challenge,
        number                      INTEGER (1..int4)
        }

getCleartextHistoricComData TRANSACTION ::= {
        ARGUMENT                    GetCleartextHistoricComDataArgument
        RESULT                      CLEAR-SECURED {HistoricComData}
        ERRORS                      {notCommissioned}
        CODE                        10
        }

GetCleartextHistoricComDataArgument ::= SEQUENCE {
        credentials                 ErtHolderCredentials,
        challenge                   Challenge,
        number                      INTEGER (1..int4)
        }

HistoricComData ::= SEQUENCE {
        number                      INTEGER (1..int4),
        outOf                       INTEGER (1..int4),
        historicRecord              CommissionErtArgument
        }
```

-- Update access control list

```
updateAccessControlList TRANSACTION ::= {
        ARGUMENT                    UpdateAccessControlListArgument
        RESULT                      NULL
        ERRORS                      {UpdateAccessControlListErrors}
        CODE                        11
        }
```

```
UpdateAccessControlListArgument ::= CHOICE {
        authorityUpdate                 SIGNED {AccessControlListUpdate, PrivateSignatureKey},
        holderUpdate                    HolderAccessControlListUpdate
        }

HolderAccessControlListUpdate ::= SEQUENCE {
        credentials                     ErtHolderCredentials,
        entry                           AccessControlListUpdate
        }

AccessControlListUpdate ::= SEQUENCE {
        mode                            ENUMERATED {deleteAllAndAdd (0), add (1), delete (2)}
                                        DEFAULT add,
        entry                           AccessControlEntry OPTIONAL
        }

AccessControlEntry ::= SEQUENCE {
        id                              EntityId,
        name                            Text OPTIONAL,
        enciphermentKeyId               KeyId OPTIONAL,
        publicEnciphermentKeyReader     PublicEnciphermentKey OPTIONAL
        }

UpdateAccessControlListErrors ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalSignature |              illegalHolderAccess |
        illegalDate |                   noEntry |
        resourceLimitExceeded |         noEnciphermentCapability |
        otherError                      }
```

-- Retrieve the access control list

```
getCiphertextAccessControlListEntry TRANSACTION ::= {
        ARGUMENT                        GetCiphertextAccessControlListEntryArgument
        RESULT                          SECURED {AuthorityAccessControlListEntry}
        CODE                            12
        }

GetCiphertextAccessControlListEntryArgument ::= SEQUENCE {
        challenge                       Challenge,
        number                          INTEGER (1..int4)
        }

getCleartextAccessControlListEntry TRANSACTION ::= {
        ARGUMENT                        GetCleartextAccessControlListEntryArgument
        RESULT                          CLEAR-SECURED {AccessControlListEntry}
        ERRORS                          {GetCleartextAccessControlListEntryErrors}
        CODE                            13
        }

GetCleartextAccessControlListEntryArgument ::= SEQUENCE {
        credentials                     ErtHolderCredentials,
        challenge                       Challenge,
        number                          INTEGER (1..int4)
        }

AuthorityAccessControlListEntry ::= AccessControlListEntry (WITH COMPONENTS {..., holderEntry ABSENT} )
```

```
AccessControlListEntry ::= SEQUENCE {
        number                          INTEGER (0..int4),
        outOf                           INTEGER (0..int4),
        authorityEntry                  AccessControlEntry OPTIONAL,
        holderEntry                     AccessControlEntry OPTIONAL
        }

GetCleartextAccessControlListEntryErrors ErrorCode ::= {
        illegalArgument |               illegalVehicleId |
        illegalHolderAccess |           otherError
        }
```

-- Get ERT capabilities

```
getErtCapabilities TRANSACTION ::= {
        ARGUMENT                        NULL
        RESULT                          ErtCapabilities
        CODE                            15
        }

ErtCapabilities ::= SEQUENCE {
        supportedTransactions           SEQUENCE OF INTEGER,
        hashingAlgorithms               SEQUENCE OF HashingAlgorithm OPTIONAL,
        signingAlgorithms               SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        signatureVerificationAlgorithms SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        secretKeyEncryptionAlgorithms   SEQUENCE OF SecretKeyAlgorithm OPTIONAL,
        publicKeyEncryptionAlgorithms   SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        publicKeyDecryptionAlgorithms   SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
        maxBitsPublicKeys               INTEGER (0..int4),
        maxOctetsPin                    INTEGER (0..int4),
        maxOctetsSetArgument            INTEGER (0..int4),
        maxNumberSetArguments           INTEGER (1..int4),
        maxNumberComArguments           INTEGER (1..int4),
        maxSizeAccessControlList        INTEGER (0..int4),
        maxNumberAuthorities            INTEGER (0..int4),
        maxNumberAddServProviders       INTEGER (0..int4),
        ertSecurityIndicationSupport    ErtSecurityFlags,
        maxInteger                      INTEGER (1..int4),
        maxStringSize                   INTEGER (1.. int4),
        …
        }
```

-- ERT tagging and authentication

```
SECURED {ToBeSecured} ::= CHOICE {
        authenticatedAndEncrypted       ENCRYPTED {ERT-AUTHENTICATED {TAGGED {ToBeSecured}}},
        authenticated                   ERT-AUTHENTICATED {TAGGED {ToBeSecured}},
        encrypted                       ENCRYPTED {TAGGED {ToBeSecured}},
        cleartext                       TAGGED {ToBeSecured}
        }

CLEAR-SECURED {ToBeSecured} ::= SECURED {ToBeSecured} (WITH COMPONENTS
        {authenticatedAndEncrypted ABSENT, encrypted ABSENT} )

TAGGED {ToBeTagged} ::= SEQUENCE {
        ertNumber                       ErtNumber,
        challenge                       Challenge OPTIONAL,
        sequenceNumber                  INTEGER (1..int4) OPTIONAL,
        tobeTagged                      ToBeTagged
        }
```

```
ERT-AUTHENTICATED {ToBeErtAuthenticated} ::= SEQUENCE {
        ertSigned                       SIGNED {ToBeErtAuthenticated, PrivateSignatureKey},
        publicVerificationKeyCertificate  ErtCertificate
        }

ErtCertificate ::= SIGNED {ErtCertificationData, PrivateSignatureKey}

ErtCertificationData ::= SEQUENCE {
        vehicleId                       VehicleId,
        publicVerificationKey           PublicVerificationKey,
        signatoryId                     EntityId,
        date                            DATE
        }
```

-- BOE Dates and Authentication

```
DATED {ToBeDated} ::= SEQUENCE {
        date                            DATE,
        validThru                       DATE OPTIONAL,
        issuer                          EntityId,
        toBeDated                       ToBeDated
        }

BOE-AUTHENTICATED {ToBeBoeAuthenticated} ::= SEQUENCE {
        signedParameter                 SIGNED {ToBeBoeAuthenticated, PrivateSignatureKey},
        certificates                    SEQUENCE SIZE(1..2) OF BoeCertificate
        }

BoeCertificate ::= SIGNED {BoeCertificationData, PrivateSignatureKey}

BoeCertificationData ::= SEQUENCE {
        entityId                        EntityId,
        entityRole                      EntityRole,
        entityName                      Text OPTIONAL,
        publicKey                       Key,
        signatoryId                     EntityId,
        signatoryName                   Text OPTIONAL,
        signatoryRole                   EntityRole,
        date                            DATE,
        validThru                       DATE
        }

EntityRole::= ENUMERATED {
        topLevelCertificationAuthority (0), intermediateCertificationAuthority (1),
        manufacturer (2),               registrationAuthority (3),
        authority (4),                  serviceProvider (5),
        eriHolder (6)                   }
```

-- Signing

```
SIGNED {ToBeSigned, PrivateSignatureKey} ::= SEQUENCE {
        toBeSigned                      ToBeSigned,
        hashingAlgorithm                HashingAlgorithm DEFAULT sha1,
        signatureAlgorithm              PublicKeyAlgorithm DEFAULT ellipticCurve,
        signature                       SIGNATURE {ToBeSigned, HashingAlgorithm,
                                        PublicKeyAlgorithm , PrivateSignatureKey}
        }
```

```
SIGNATURE {ToBeSigned, HashingAlgorithm, SignatureAlgorithm, PrivateSignatureKey} ::= BIT STRING
        (CONSTRAINED BY
{HashingAlgorithm, -- and -- SignatureAlgorithm, -- with -- PrivateSignatureKey, -- applied to -- ToBeSigned} )

Challenge ::= INTEGER (1..int4)

-- Encryption

ENCRYPTED {ToBeErtEncrypted} ::= SEQUENCE {
        enciphermentKeyId               KeyId,
        publicKeyEncryptionAlgorithm    PublicKeyAlgorithm DEFAULT ellipticCurve,
        secretKeyEncryptionAlgorithm    SecretKeyAlgorithm DEFAULT aes,
        encryptedKey                    KEY-ENCRYPTION {SecretTransactionKey,
                                        PublicKeyAlgorithm, PublicEnciphermentKey},
        ciphertext                      CIPHER-TEXT {ToBeErtEncrypted,
                                        SecretKeyAlgorithm, SecretTransactionKey}
        }

KEY-ENCRYPTION {KeyToBeEncrypted, PublicKeyAlgorithm, PublicEnciphermentKey} ::=
        BIT STRING ( CONSTRAINED BY {
        PublicKeyAlgorithm, -- with -- PublicEnciphermentKey, -- applied to -- KeyToBeEncrypted
        -- or random bit string with same length if no public key is available --} )

CIPHER-TEXT {ToBeEncrypted, SecretKeyAlgorithm, SecretKey} ::=
        BIT STRING ( CONSTRAINED BY {
        SecretKeyAlgorithm, -- with -- SecretKey, -- applied to -- ToBeEncrypted
        -- or random bit string with same length if no public key is available --} )

-- Encryption algorithms

HashingAlgorithm ::= ENUMERATED {
        sha1,
        ... }

PublicKeyAlgorithm  ::= ENUMERATED {
        ellipticCurve,
        ... }

SecretKeyAlgorithm  ::= ENUMERATED {
        aes,
        ... }

-- Credentials and keys

ErtHolderCredentials ::= SEQUENCE {
        vehicleId                       VehicleId,
        pin                             PIN
        }

SecretTransactionKey                    ::= Key
PublicEnciphermentKey                   ::= Key
PrivateDeciphermentKey                  ::= Key
PrivateSignatureKey                     ::= Key
PublicVerificationKey                   ::= Key
Key                                     ::= BIT STRING

PIN                                     ::= NumericString (SIZE(4))

KeyId                                   ::= INTEGER (0..65535)
```