International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.510

(08/2020)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

Directory

Information technology – Open Systems Interconnection – The Directory: Protocol specifications for secure operations

Recommendation ITU-T X.510

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.379
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	**X.500–X.599**
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems management framework and architecture	X.700–X.709
Management communication service and protocol	X.710–X.719
Structure of management information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, concurrency and recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.889
Generic applications of ASN.1	X.890–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	X.1000–X.1099
SECURE APPLICATIONS AND SERVICES (1)	X.1100–X.1199
CYBERSPACE SECURITY	X.1200–X.1299
SECURE APPLICATIONS AND SERVICES (2)	X.1300–X.1499
CYBERSECURITY INFORMATION EXCHANGE	X.1500–X.1599
CLOUD COMPUTING SECURITY	X.1600–X.1699
QUANTUM COMMUNICATION	X.1700–X.1729
DATA SECURITY	X.1750–X.1799
5G SECURITY	X.1800–X.1819

For further details, please refer to the list of ITU-T Recommendations.

INTERNATIONAL STANDARD ISO/IEC 9594-11
RECOMMENDATION ITU-T X.510

Information technology – Open Systems Interconnection – The Directory: Protocol specifications for secure operations

Summary

Recommendation ITU-T X.510 | ISO/IEC 9594-11 specifies a general protocol, called the wrapper protocol, that provides cybersecurity for protocols designed for its protection. The wrapper protocol provides authentication, integrity and optionally confidentiality (encryption). The wrapper protocol allows cybersecurity to be provided independently of the protected protocols, which means that security may be enhanced without affecting protected protocol specifications.

The wrapper protocol is specified without specifying specific cryptographic algorithms, but is designed for plucking-in cryptographic algorithms as required.

The wrapper protocol is designed for easy migration of cryptographic algorithms, as stronger cryptographic algorithms become necessary.

Recommendation ITU-T X.510 | ISO/IEC 9594-11 contains recommendations for how other Recommendations and International Standards may include features for migration of cryptographic algorithms, and it includes ASN.1 specifications to be applied for that purpose.

Recommendation ITU-T X.510 | ISO/IEC 9594-11 also specifies three protocols that make use of the wrapper protocol protection. This includes a protocol for maintenance of authorization and validation lists (AVLs), a protocol for subscribing of public-key certificate status and a protocol for accessing a trust broker.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T X.510	2020-08-22	17	11.1002/1000/14320

Keywords

Certification authority, cryptography, cryptographic algorithm, digital signature, public-key certificate, PKI, quantum-safe, trust anchor, validation.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

Introduction

The Internet Engineering Task Force (IETF) maintains a substantial set of protocols for supporting public-key infrastructure (PKI). Recommendation ITU-T X.510 | ISO/IEC 9594-11 provides protocols to supplement those protocols developed by IETF, especially for:

a) supporting new functions specified by Rec. ITU-T X.509 | ISO/IEC 9594-8, for which IETF has not provided support, e.g., support for authorization and validation list (AVL) maintenance;

b) constraint environments, where lean protocols are required.

In addition, it specifies:

c) a wrapper protocol that provides security services for other protocols.

This Recommendation | International Standard consist of three sections as follows.

Section 1 gives general specifications for this Recommendation | International Standard.

Section 2 is the wrapper protocol specification.

Section 3 specifies some protocols to be protected by the wrapper protocol:

a) a protocol for maintaining authorization and validation lists (AVLs);

b) a protocol for subscribing public-key certificate status information from certification authorities (CAs); and

c) a protocol for accessing a trust broker.

The following annexes are included.

Annex A, which is an integral part of this Recommendation | International Standard, provides the ASN.1 module for specifications to be imported by protocols providing a migration path for cryptographic algorithms.

Annex B, which is an integral part of this Recommendation | International Standard, provides the ASN.1 module for the wrapper protocol.

Annex C, which is an integral part of this Recommendation | International Standard, provides specifications for how a protected protocol is wrapped by the wrapper protocol.

Annex D, which is an integral part of this Recommendation | International Standard, provides cryptographic algorithm specification.

Annex E, which is an integral part of this Recommendation | International Standard, provides the ASN.1 module for maintenance of the authorization and validation lists (AVLs) protocol.

Annex F, which is an integral part of this Recommendation | International Standard, provides the ASN.1 module for certification authority subscription protocol.

Annex G, which is an integral part of this Recommendation | International Standard, provides the ASN.1 module for the trust broker protocol.

Annex H, which is not an integral part of this Recommendation | International Standard, provides guidance for cryptographic algorithm migration.

The content of this Rec. ITU-T X.510 | ISO/IEC 9594-11 was moved to here from Rec. ITU-T X.509 (2016) | ISO/IEC 9594-8:2017 and subsequently updated.

INTERNATIONAL STANDARD ISO/IEC 9594-11
RECOMMENDATION ITU-T X.510

Information technology – Open Systems Interconnection – The Directory: Protocol specifications for secure operations

SECTION 1 – GENERAL

1 Scope

The scope of this Recommendation | International Standard is threefold.

This Recommendation | International Standard provides guidance on how to prepare new and old protocols for cryptographic algorithm migration, and defines auxiliary cryptographic algorithms to be used for migration purposes.

This Recommendation | International Standard specifies a general wrapper protocol that provides authentication, integrity and confidentiality (encryption) protection for other protocols. This wrapper protocol includes a migration path for cryptographic algorithms allowing for smooth migration to stronger cryptographic algorithms as such requirements evolve. This will allow migration to quantum-safe cryptographic algorithms. Protected protocols can then be developed without taking security and cryptographic algorithms into consideration.

This Recommendation | International Standard also includes some protocols to be protected by the wrapper protocol primarily for support of public-key infrastructure (PKI). Other specifications, e.g., Recommendations or International Standards, may also develop protocols designed to be protected by the wrapper protocol.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.500 (2019) | ISO/IEC 9594-1:2020, *Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services.*
- Recommendation. ITU-T X.501 (2019) | ISO/IEC 9594-2:2020, *Information technology – Open Systems Interconnection – The Directory: Models.*
- Recommendation ITU-T X.509 (2019) | ISO/IEC 9594-8:2020, *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks.*
- Recommendation ITU-T X.511 (2019) | ISO/IEC 9594-3:2020, *Information technology - Open Systems Interconnection - The Directory: Abstract service definition.*
- Recommendation ITU-T X.518 (2019) | ISO/IEC 9594-4:2020, *Information technology - Open Systems Interconnection - The Directory: Procedures for distributed operation.*
- Recommendation ITU-T X.519 (2019) | ISO/IEC 9594-5:2020, *Information technology - Open Systems Interconnection - The Directory: Protocol specifications.*
- Recommendation ITU-T X.520 (2019) | ISO/IEC 9594-6:2020, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types.*
- Recommendation ITU-T X.521 (2019) | ISO/IEC 9594-7:2020, *Information technology - Open Systems Interconnection - The Directory: Selected object classes.*
- Recommendation ITU-T X.525 (2019) | ISO/IEC 9594-9:2020, *Information technology - Open Systems Interconnection - The Directory: Replication.*
- Recommendation ITU-T X.680 (2015)| ISO/IEC 8824-1:2015, *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

- Recommendation ITU-T X.681 (2015) | ISO/IEC 8824-2:2015, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- Recommendation ITU-T X.682 (2015) | ISO/IEC 8824-3:2015, *Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification*.
- Recommendation ITU-T X.683 (2015) | ISO/IEC 8824-4:2015, *Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
- Recommendation ITU-T X.690 (2015) | ISO/IEC 8825-1:2015, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*
- Recommendation ITU-T X.691 (2015) | ISO/IEC 8825-2:2015, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).*

2.2 Paired Recommendations | International Standards equivalent in technical content

- Recommendation ITU-T X.800 (1991), *Security architecture for Open Systems Interconnection for CCITT applications*.

  ISO 7498-2:1989, *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture*.

2.3 Other references

- IETF RFC 793 (1981), *Transmission Control Protocol*.
- IETF RFC 2104 (1997), *HMAC: Keyed-Hashing for Message Authentication*.
- IETF RFC 3526 (2003), *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE).*
- IETF RFC 5084 (2007), *Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS).*
- IETF RFC 5114 (2008), *Additional Diffie-Hellman Groups for Use with IETF Standards*.
- IETF RFC 5869 (2010), *HMAC-based Extract-and-Expand Key Derivation Function (HKDF).*
- IETF RFC 6932 (2013), *Brainpool Elliptic Curves for the Internet Key Exchange (IKE) Group Description Registry*.

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply:

3.1 OSI Reference Model definitions

The following terms are defined in Rec. ITU-T X.800 | ISO 7498-2:

a) confidentiality;

b) cryptography;

c) digital signature.

3.2 Directory model definitions

The following terms are defined in Rec. ITU-T X.501 | ISO/IEC 9594-2:

a) attribute;

b) distinguished name (of an entry).

3.3 Public-key and attribute certificate definitions

The following terms are defined in Rec. ITU-T X.509 | ISO/IEC 9594-8:

a) authorization and validation list (AVL);

b) authorization and validation list entity (AVL entity);

c) authorizer;

d) certification authority (CA);
e) certification path;
f) end entity;
g) end-entity public-key certificate;
h) hash function;
i) key agreement;
j) private key;
k) public key;
l) public-key certificate;
m) public-key infrastructure (PKI);
n) relying party;
o) trust broker.

3.4 Terms defined in this Recommendation | International Standard

3.4.1 abstract syntax: A specification of application-protocol-data-units by using notation rules that are independent of the encoding technique used to represent them.

NOTE – The term abstract syntax is original an OSI term but is extended here to be general applicable.

3.4.2 alternative cryptographic algorithm: A cryptographic algorithm to which migration is wanted.

3.4.3 application entity: An active element embodying a set of capabilities which is pertinent to communication systems and which is defined for the application layer.

NOTE – The term application entity is originally an OSI term (see Rec. ITU-T X.519 | ISO/IEC 9594-5), but is extended here to be generally applicable.

3.4.4 application protocol data unit (APDU): Data that is transmitted as a single unit at the application layer between two application entities.

3.4.5 association: A cooperative relationship between two application entities, which enables the communication of information and the coordination of their joint operation for an instance of communication.

3.4.6 client: The entity that initiates an association.

3.4.7 data transfer phase: The phase from the completion of the establishment of an association to the termination of the association.

3.4.8 digital signature: The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for origin authentication, data integrity and signatory non-repudiation.

3.4.9 native cryptographic algorithm: A cryptographic algorithm used prior to a migration period.

3.4.10 protected protocol data unit (PrPDU): Application protocol data unit defined by an application protocol to be protected by the wrapper protocol.

3.4.11 server: The entity that accepts or rejects an association.

3.4.12 symmetric key: A cryptographic key used for both encryption of plaintext and decryption of ciphertext.

3.4.13 wrapper protocol data unit (WrPDU): An application protocol data unit carrying security protocol control information and, when relevant, carrying a protected protocol data unit.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-CBC	Advanced Encryption Standard-Cipher Block Chaining
APDU	Application Protocol Data Unit
ASN.1	Abstract Syntax Notation One

AVL	Authorization and Validation List
AVMP	Authorization and Validation Management Protocol
BER	Basic Encoding Rules
CA	Certification Authority
CASP	Certification Authority Subscription Protocol
DER	Distinguished Encoding Rules
DH	Diffie-Hellman
HKDF	HMAC-based extract-and-expand Key Derivation Function
HMAC	keyed-Hash Message Authentication Code
ICV	Integrity Check Value
ICT	Information and Communications Technology
ID	Identifier
LoA	Loss of Alignment
MODP	Modular exponential
OKM	Output Keying Material
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PKI	Public-Key Infrastructure
PMI	Privilege Management Infrastructure
PRK	Pseudorandom Key
PrPDU	Protected protocol Data Unit
RAOC	Receive Application Object Class
SAOC	Send Application Object Class
TCP	Transmission Control Protocol
UTC	Coordinated Universal Time
WrPDU	Wrapper Protocol Data Unit

5 Conventions

The term "Specification" (as in "this Specification") shall be taken to mean this Recommendation | International Standard.

If an International Standard or ITU-T Recommendation is referenced within normal text without an indication of the edition, the edition shall be taken to be the one specified in the normative references clause.

This Specification makes extensive use of the abstract syntax notation one (ASN.1) for the formal specification of data types and values, as it is specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2, Rec. ITU-T X.682 | ISO/IEC 8824-3, Rec. ITU-T X.683 | ISO/IEC 8824-4, Rec. ITU-T X.690 | ISO/IEC 8825-1 and Rec. ITU-T X.691 | ISO/IEC 8825-2.

This Specification presents ASN.1 notation in the **bold Courier New** typeface. When ASN.1 types and values are referenced in normal text, they are differentiated from normal text by presenting them in the **bold Courier New** typeface.

6 Common data types and special cryptographic algorithms

6.1 Introduction

This clause defines some auxiliary cryptographic specification as follows.

a) ASN.1 information object classes are heavily used for protocol design. The **ALGORITHM** information object class is important for this Specification. This is further expanded in clause 6.2.

b) Multiple cryptographic algorithms of a specific class may be specified by using a single containing cryptographic algorithm. This is done by utilizing the flexibility provided by the **AlgorithmIdentifier**

parameterized data type defined in clause 6.2.2 of Rec. ITU-T X.509 | ISO/IEC 9594-8. This is further described in clause 6.3.

c) There are advantages to the definition of cryptographic algorithms for key agreement procedures. This has been done for a few cases in clause 6.4.

d) Parameterized data types are defined in clause 6.5 for flexible protocol design.

e) Some formal specifications for encipherment are given in clause 6.6.

6.2 ASN.1 information object class specification tool

6.2.1 General information object class concept

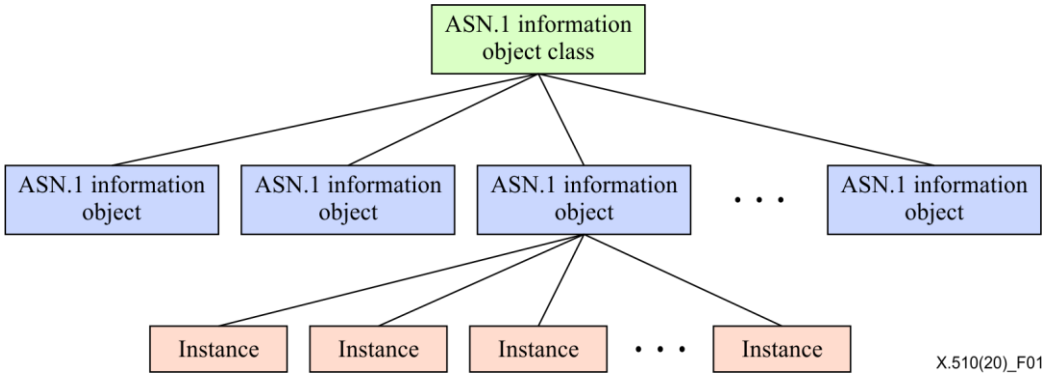

Figure 1 – ASN.1 information object class concept

The concept of ASN.1 information object class is specified in Rec. ITU-T X.681 | ISO/IEC 8824-2. It is vital for protocol design. For that reason, a short introduction is included here to encourage increased use of this facility. ASN.1 information object classes are widely used by the ITU-T X.500 series of Recommendations | ISO/IEC 9594-all parts for defining attributes, matching rules, extensions to public-key and attribute certificates, etc.

Figure 1 illustrates the general ASN.1 information object class concept. An ASN.1 information object class specifies the general syntax for a class of information objects, e.g., attribute types as defined by Rec. ITU-T X.501 | ISO/IEC 9594-2. From this general specification, specifications for specific attribute types are defined, e.g., an attribute type for e-mail addresses. From this specification, instances of e-mail address attributes may be generated. Instances may be transferred in the protocol or may be stored in a directory.

NOTE – The concept of object classes used in clause 11 is somewhat different from the concept of information object class defined in Rec. ITU-T X.681 | ISO/IEC 8824-2.

6.2.2 The ALGORITHM information object class

The information object class concept is also used to define cryptographic algorithms. The **ALGORITHM** information object class is defined in 6.2.2 of Rec. ITU-T X.509 | ISO/IEC 9594-8. The specification of this information is reproduced as follows for easy reference. The **ALGORITHM** information object class is different from most other information object classes in the sense that an instance of an information object is an invocation of the algorithm rather than specifying a value identifying something, like an e-mail address.

The following ASN.1 information object class is used to specify cryptographic algorithms.

```
ALGORITHM ::= CLASS {
  &Type           OPTIONAL,
  &DynParms       OPTIONAL,
  &id             OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  [PARMS          &Type]
  [DYN-PARMS      &DynParms ]
  IDENTIFIED BY   &id }
```

The **ALGORITHM** information object class has the following fields.

a) The **&Type** field is used to specify those fixed parameters that are necessary for specifying the exact procedure for deploying the cryptographic algorithm being defined. Not all cryptographic algorithms

require such parameters. The field is then absent or has the value **NULL**, as determined by the individual cryptographic algorithm specifications.

b) The **&DynParms** field is used to specify those dynamic parameters that determine the value(s) to be exchanged between two communicating entities when invoking the cryptographic algorithm. Not all cryptographic algorithms require dynamic parameters. In this case the **&DynParms** field shall be absent.

c) The **&id** field is used to uniquely identify the class of cryptographic algorithm being defined.

The **AlgorithmWithInvoke** parameterized data type defined as follows is used in situations where the type of cryptographic algorithm is signalled together with its invocation.

```
AlgorithmWithInvoke{ALGORITHM:SupportedAlgorithms} ::= SEQUENCE {
  algorithm       ALGORITHM.&id({SupportedAlgorithms}),
  parameters  [0] ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL,
  dynamParms  [1] ALGORITHM.&DynParms({SupportedAlgorithms}{@algorithm}) OPTIONAL,
  ... }
```

The **AlgorithmWithInvoke** parameterized data type has the following components.

a) The **algorithm** component shall hold the object identifier that uniquely identify the cryptographic algorithm being defined.

b) The **parameters** component, when present, shall hold the values of the fixed parameters that further identify the cryptographic algorithm in question. This component shall be present when the **&Type** field is present in the information object for the cryptographic algorithm in question. Otherwise, it shall be absent.

c) The **dynamParms** component, when present, shall hold the value(s) required by the dynamic parameters for the cryptographic algorithm. This component shall be present when the **&DynParms** field is present in the information object for the cryptographic algorithm. Otherwise, it shall be absent.

The **AlgorithmIdentifier** parameterized data type defined as follows is used in situations where the type of cryptographic algorithm is signalled without a corresponding invocation.

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

The components of **AlgorithmIdentifier** data type shall be as specified for the corresponding components of the **AlgorithmWithIvoke** parameterized data type.

The **AlgoInvoke** parameterized data type defined as follows is used when the cryptographic algorithm has previously been determined and where only invocation information is required.

```
AlgoInvoke{ALGORITHM:SupportedAlgorithms} ::=
    ALGORITHM.&DynParms({SupportedAlgorithms})
```

6.3 Multiple-cryptographic algorithm specifications

6.3.1 General

Multiple cryptographic algorithms of the same object class may be specified using a single outer algorithm and then used instead of a single algorithm as a tool for algorithm migration as discussed in Annex H.

The **PARMS** field of the **ALGORITHM** information object class allows any data type to be specified. This is utilized to define **ALGORITHM** information objects that allow for multiple-cryptographic algorithm specifications within a single algorithm specification.

6.3.2 Multiple signatures algorithm

The following is a specification of an **ALGORITHM** information object that allows multiple digital signature algorithms to be specified.

```
multipleSignaturesAlgo ALGORITHM ::= {
  PARMS         MultipleSignaturesAlgo
  IDENTIFIED BY id-algo-multipleSignaturesAlgo }

MultipleSignaturesAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedSignatureAlgorithms}}
```

```
SupportedSignatureAlgorithms ALGORITHM ::= {...}
```

6.3.3 Multiple symmetric key algorithm

The following is a specification of an **ALGORITHM** information object that allows multiple symmetric key algorithms to be specified.

```
multipleSymmetricKeyAlgo ALGORITHM ::= {
  PARMS         MultipleSymmetricKeyAlgo
  IDENTIFIED BY id-algo-multipleSymmetricKeyAlgo }

MultipleSymmetricKeyAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedSymmetricKeyAlgorithms}}

SupportedSymmetricKeyAlgorithms ALGORITHM ::= {...}
```

6.3.4 Multiple public-key algorithms

The following is a specification of an **ALGORITHM** information object that allows multiple public-key algorithms to be specified.

```
multiplePublicKeyAlgo ALGORITHM ::= {
  PARMS         MultiplePublicKeyAlgo
  IDENTIFIED BY id-algo-multiplePublicKeyAlgo }

MultiplePublicKeyAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedPublicKeyAlgorithms}}

SupportedPublicKeyAlgorithms ALGORITHM ::= {...}
```

6.3.5 Multiple hash algorithm

The following is a specification of an **ALGORITHM** information object that allows multiple hash algorithms to be specified.

```
multipleHashAlgo ALGORITHM ::= {
  PARMS         MultipleHashAlgo
  IDENTIFIED BY id-algo-multipleHashAlgo }

MultipleHashAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedHashAlgorithms}}

SupportedHashAlgorithms ALGORITHM ::= {...}
```

6.3.6 Multiple authenticated encryption with associated data algorithm

The following is a specification of an **ALGORITHM** information object that allows multiple authenticated encryption with associated data (AEAD) algorithms to be specified.

```
multipleAuthenEncryptAlgo ALGORITHM ::= {
  PARMS         MultipleAuthenEncryptAlgo
  IDENTIFIED BY id-algo-multipleAuthenEncryptAlgo }

MultipleAuthenEncryptAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo       AlgorithmIdentifier{{SupportedAuthenEncryptAlgorithms}}

SupportedAuthenEncryptAlgorithms ALGORITHM ::= {...}
```

6.3.7 Multiple integrity check value algorithm

The following is a specification of an **ALGORITHM** information object that allows multiple integrity check value (ICV) algorithms to be specified.

```
multipleIcvAlgo ALGORITHM ::= {
  PARMS         MultipleIcvAlgo
  IDENTIFIED BY id-algo-multipleIcvAlgo }

MultipleIcvAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedIcvAlgorithms}}

SupportedIcvAlgorithms ALGORITHM ::= {...}
```

6.4 Key establishment algorithms

6.4.1 General

Key establishment technologies are used to establish symmetric keys directly between two communicating entities in a secure way. The Diffie-Hellman (DH) technology, as specified in IETF RFC 2631, is an important example. The result of a DH operation is a shared secret that a key derivation technology expands to generate the required symmetric keys.

DH and the key derivation technologies in their basic form are not well suited to the cryptographic algorithm migration techniques described in Annex H. By defining different combination of DH and key derivation combinations as cryptographic algorithms using the **ALGORITHM** information object class, as it specified in clause 6.2.2, it is possible to establish a migration path. In the following, some key establishment information objects are defined.

Clause 6.4 is provided for the purpose of validating the wrapper protocol as a whole using DH technology, as specified in IETF RFC 2631. The choice of the cryptographic mechanism for the shared key generation in the wrapper protocol is made by end-users in accordance with the principles of national regulations.

6.4.2 Diffie-Hellman group 14 algorithm with HKDF-256

The following is a specification for key establishment algorithm based on the DH key agreement technique.

```
dhModpGr14Hkdf256Algo ALGORITHM ::= {
  PARMS         Group14
  DYN-PARMS     Payload14
  IDENTIFIED BY id-algo-dhModpGr14Hkdf256Algo }

Group14 ::= INTEGER (14)

Payload14 ::= SEQUENCE {
  dhPublicKey OCTET STRING (SIZE (256)),
  nonce       OCTET STRING (SIZE (32)),
  ... }
```

The **PARMS** token specifies that the fixed parameters shall be those parameters specified for group number 14 for the 2048-bit modular exponential (MODP*)* as specified in IETF RFC 3526.

The **DYN-PARMS** token specifies that the dynamic parameters shall be those specified by the **PayLoad14** data type.

The **PayLoad14** data type has the following components.

a) The **dhPublicKey** component shall hold the DH public key to be used by the sender. For the group used, the length of the key is always 256 octets. A different DH private and public key pair shall be generated for each new association establishment. For the duration of the association, the DH public key of the server shall be retained by both parties and the server shall retain its DH private key for later renewal of symmetric keys (see clause 10.2).

b) For each key renewal, the client shall generate a new DH private and DH public key pair and use it for the key renewal process.

c) The **nonce** component shall hold a random value to be used by the shared key derivation specified in clause 6.4.5. A new value shall be generated for each key establishment. The length is recommended to be that of the hash output. As the **sha256** algorithms is used for key derivation, the length shall be 32 octets.

From the established shared secret, the two partners shall use the HMAC-based extract-and-expand key derivation function (HKDF) to create symmetric keys according to the requirement established by the encryption and ICV negotiation.

The **hmacWithSHA256** algorithm shall be used for the HKDF derivation.

6.4.3 Diffie-Hellman group 23 algorithm with HKDF-256

```
dhModpGr23Hkdf256Algo ALGORITHM ::= {
  PARMS         Group23
  DYN-PARMS     Payload23
  IDENTIFIED BY id-algo-dhModpGr23Hkdf256Algo }

Group23 ::= INTEGER (23)

Payload23 ::= SEQUENCE {
  dhPublicKey OCTET STRING (SIZE (512)),
  nonce       OCTET STRING (SIZE (32)),
```

```
  ... }
```

The **PARMS** token specifies that the fixed parameters shall be those parameters specified for group number 23, for the ECDH curve **secp256r1** as specified in IETF RFC 5114.

The **DYN-PARMS** token specifies that the dynamic parameters shall be those specified by the **PayLoad23** data type.

The **PayLoad23** data type has the same components as the **PayLoad14** data type except that the **dhPublicKey** shall have a length of 512 octets.

The key derivation shall be as specified in clause 6.4.5.

6.4.4 Diffie-Hellman group 28 algorithm with HKDF-256

```
dhModpGr28Hkdf256Algo ALGORITHM ::= {
  PARMS          Group28
  DYN-PARMS      Payload28
  IDENTIFIED BY id-algo-dhModpGr28Hkdf256Algo }

Group28 ::= INTEGER (28)

Payload28 ::= SEQUENCE {
  dhPublicKey OCTET STRING (SIZE (512)),
  nonce       OCTET STRING (SIZE (32)),
  ... }
```

The **PARMS** token specifies that the fixed parameters shall be those parameters specified for group number 28, for the ECDH curve **sbrainpoolP256r1** as specified in IETF RFC 6932.

The **DYN-PARMS** token specifies that the dynamic parameters shall be those specified by the **PayLoad28** data type.

The **PayLoad28** data type has the same components as the **PayLoad14** data type, except that the **dhPublicKey** shall have a length of 512 octets.

The key derivation shall be as specified in clause 6.4.5.

6.4.5 Key derivation

6.4.5.1 General

When the outcome of a key agreement algorithm is a shared secret, this shared secret shall be expanded by use of a key derivation function to provide sufficient material for the required symmetric keys.

6.4.5.2 HMAC-based extract-and-expand key derivation function

The HKDF is specified in IETF RFC 5869. It requires the use of the keyed-hash message authentication code (HMAC) algorithm as specified in IETF RFC 2104.

The HKDF consists of an HKDF-extract, resulting in a so-called pseudorandom key (PRK) and of an HKDF-expand, resulting in the output keying material (OKM).

The HKDF-extract may be expressed as follows.

PRK = HMAC-Hash(salt, IKM) where:

a) **salt** is a non-secret random value – it shall take the value of the **nonce** component of the key establishment algorithm instance in question that require the use of HKDF;

b) **IKM** stands for input keying material – it shall be the shared secret resulting from the DH key agreement;

c) The **OKM** is defined as:

OKM = HKDF-expand (PRK, info, L), where:

- **PRK** shall be the **PRK** generated by the **HKDF-expand** above,
- **info** shall be a zero-length string,
- **L** shall be the combined length of the keys to be generated;

If two keys are to be generated:

- the first part of the **OKM** is the key to be used by the client,
- the remaining part is the key to be used by the server;

If four keys are to be generated:

- the first key is the key used to generate an ICV by the client,
- the second key is the key used to generate an ICV by the server,
- the third key is the key used for encryption by the client,
- the fourth key is the key used for encryption by the server.

6.4.6 Special conditions

NOTE – For authenticated key agreement protocols, the absence of an explicit key validation phase [should] be compensated for by mixing up the certificates of both parties when generating the shared key. Such mixing up is not carried out in this protocol, which may lead to undesired consequences.

6.5 Multiple-cryptographic algorithm-value pairs

6.5.1 Multiple digital signatures attached to data

The **MULTI_SIGNED** parameterized data type is an expansion of the **SIGNED** data type defined in clause 6.2.1 of Rec. ITU-T X.509 | ISO/IEC 9594-8. It allows for multiple signatures to be attached to data to be digitally signed.

```
MULTY-SIGNED{ToBeSigned} ::= SEQUENCE {
  toBeSigned  ToBeSigned,
  algorithm   ALGORITHM.&id({multipleSignaturesAlgo}),
  parmeters   SEQUENCE SIZE (1..MAX) OF
    sign        SEQUENCE {
      algo        AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
      signature   BIT STRING,
      ... },
  ... }
```

Depending on the deployment, it may be required that all signatures be verified. This requires that the recipient supports all the involved digital signature algorithms. In other cases, a single signature is sufficient for verification.

6.5.2 Double digital signature attached to data

The **Signed** parameterized data type is used instead of the **SIGNED** data type, when signalling of digital signature algorithms is not necessary.

```
Signed{ToBeSigned} ::= SEQUENCE {
  toBeSigned   ToBeSigned,
  signature    BIT STRING,
  altSignature BIT STRING OPTIONAL,
  ... }
```

The **Signed** data type has the following components:

a) the **toBeSigned** component shall hold the value of the data type to be signed;

b) the **signature** component shall hold what is called the native digital signature to be attach to the data – the digital signature shall be generated according to a digital signature algorithm that has previously been signalled;

c) the **altSignature** component may be present if an alternative digital signature algorithm previously has been signalled – otherwise, it shall be absent.

6.5.3 Duplicate integrity check values attached to data

The **ICV** parameterized data types are counterparts to the **SIGNED** parameterized data type as defined in clause 6.2.1 of Rec. ITU-T X.509 | ISO/IEC 9594-8. They provide for attaching one or two ICVs to data whose integrity is to be protected.

The **ICV-Total** parameterized data type is used where the ICV algorithm has not previously been agreed between two communication entities.

```
ICV-Total{ToBeProtected} ::= SEQUENCE {
  toBeProtected               ToBeProtected,
  algorithmIdentifier         AlgorithmWithInvoke{{SupportedIcvAlgorithms}},
  icv                         OCTET STRING,
  altAlgorithmIdentifier  [0] AlgorithmWithInvoke{{SupportedIcvAlgorithms}} OPTIONAL,
  altIcv                  [1] OCTET STRING OPTIONAL,
  ... }
   (WITH COMPONENTS {..., altAlgorithmIdentifier PRESENT, altIcv PRESENT } |
    WITH COMPONENTS {..., altAlgorithmIdentifier ABSENT,  altIcv ABSENT } )
```

The **ICV-Total** data type has the following components.

a) The **toBeProtected** component shall specify the data value of what is passed in a parameter, being the data value to be protected.

b) The **algorithmIdentifier** component consists of the following components (see clause 6.2.2):

– the **algorithm** component shall hold the object identifier assigned to the ICV algorithm in question;

– the **parameters** component, when present, shall hold the value of the fixed parameters – this component shall be present when the **&Type** field is present in the information object for the ICV algorithm in question – otherwise, it shall be absent;

– The **dynamParms** component, when present, shall hold the value of the dynamic parameters – this component shall be present when the **&DynParms** field is present in the information object for the ICV algorithm in question – otherwise, it shall be absent.

c) The **icv** component shall hold the value of the generated ICV using the ICV algorithm identified by the **algorithm** and **parameters** components listed in b) and by applying the **dynamParms** component, if relevant.

d) The **altAlgorithmIdentifier** component, when present, consists of the same components as the **algorithmIdentifier** component.

e) The **altIcv** component, when present, shall hold the value of the generated ICV using ICV algorithm identified by the **algorithm** and **parameters** components of the **altAlgorithmIdentifier** component.

The **altAlgorithmIdentifier** and **altIcv** components shall either both be present, or both be absent.

The **ICV-Invoke** parameterized data type is used where the ICV algorithm has previously been agreed between two communication entities, e.g., during a handshake.

```
ICV-Invoke{ToBeProtected} ::= SEQUENCE {
  toBeProtected      ToBeProtected,
  dynParms       [0] AlgoInvoke{{SupportedIcvAlgorithms}} OPTIONAL,
  icv                BIT STRING,
  ... }
```

The **ICV-Total** data type has the following components.

a) The **toBeProtected** component shall specify the data value of what is passed in a parameter, being the data value to be protected.

b) The **dynParms** component, when present, shall hold the value of the dynamic parameters associated with the ICV algorithm in question. This component shall be present when the **&DynParms** field is present in the information object for that ICV algorithm. Otherwise, it shall be absent.

c) The **icv** component shall hold the generated ICV using the previously agreed ICV algorithm and, if relevant, by applying the **dynamParms** component.

6.6 Formal specification of encipherment

6.6.1 Formal specification of encryption

```
ENCIPHERED{ToBeEnciphered} ::= OCTET STRING (CONSTRAINED BY {
   -- shall be the result of applying an encipherment procedure
   -- to the encoded octets of a value of -- ToBeEnciphered } )
```

6.6.2 Formal specification of authenticated encryption with associated data

The following parameterized data type is used to specify AEAD.

```
AUTHEN-ENCRYPT{ToBeAuth, ToBeEnciphered} ::= SEQUENCE {
  aad  ToBeAuth,
  encr ToBeEnciphered,
  ... }
```

IETF RFC 5084 is a specification for the use of AEAD. IETF RFC 5084 has associated data as optional, while it is mandatory in this Specification.

7 General concepts for securing protocols

7.1 Introduction

Protocols used in information and communications technology (ICT) infrastructures need to include cybersecurity capabilities to make them resistant to attacks. The intention of this Specification is to provide a structure where the cybersecurity aspects are clearly separated from the actual protocol requiring cybersecurity. This is done by a specification of a general protocol, called the wrapper protocol, which includes all the cybersecurity aspects. This wrapper protocol can then embed another protocol, called the protected protocol, and in this way supply cybersecurity to the embedded or wrapped protocol. A protocol that is structured according to this Specification can be protected by the wrapper protocol and needs no security capabilities of its own.

The wrapper protocol may protect protocols specific for maintenance of a PKI or a privilege management infrastructure (PMI), but may also be used for protocols with different purposes.

7.2 Protected protocol plug-in concept

Figure 2 is one way of illustrating the relationship between the wrapper protocol and a protected protocol. A protected protocol is seen as embedded in the wrapper protocol, which then provides a protecting shield for the protected protocol. The wrapper protocol can protect any protocol that has been designed for protection by the wrapper protocol (see clauses 7.4 and 8.10).

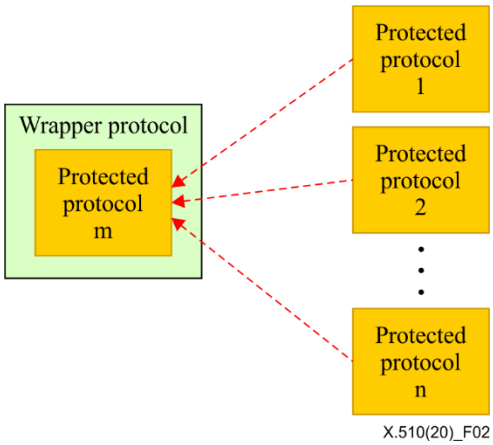

Figure 2 – Protected protocol plug-in

7.3 Communications structure

Figure 3 depicts the communications between two entities that are each running the wrapper protocol protecting an instance of the same protocol. The data exchange by the protected protocol is then protected by the two instances of the wrapper protocol. The two instances of the wrapper protocol may also need to communicate without involving the instances of the protected protocol.

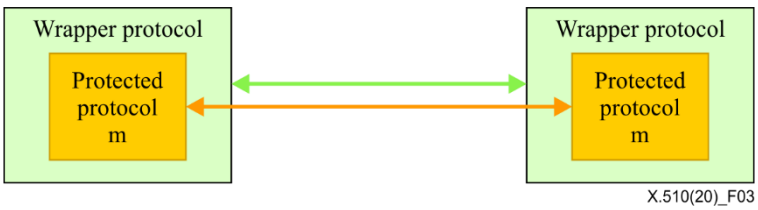

Figure 3 – Embedded communication

7.4 Another view of the relationship between the wrapper protocol and the protected protocol

Figure 4 illustrates how the wrapper protocol embeds a protected protocol. At different places within the wrapper protocol, it interacts with the protected protocol. As the protected protocol specification includes several protected protocol data unit (PrPDU) types, it is required that the top statement of the protected protocol is a choice among the defined PrPDU types.

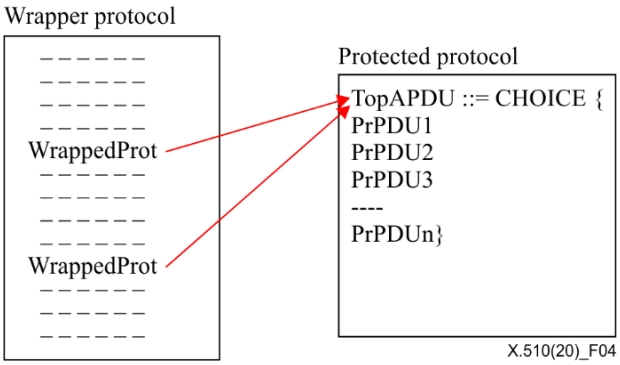

Figure 4 – Inclusion of the protected protocol

7.5 Structure of application protocol data unit

Figure 5 depicts the structure of the transmitted application protocol data unit (APDU), called the wrapper protocol data unit (WrPDU). When actual data is to be transmitted by the two instances of the protected protocol, this data it is embedded in the WrPDU in the form of a protected protocol data unit (PrPDU). When the two instances of the wrapper protocol need to interact without involving the protected protocol, there will be no embedded PrPDU.

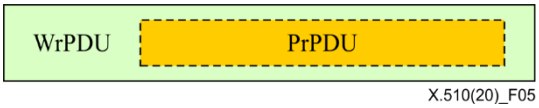

Figure 5 – APDU structure

7.6 Exception conditions

This Specification requires checks of received APDUs (WrPDUs and PrPDUs). The objective is to force compliant implementations to perform these checks to ensure resilient implementations.

This Specification defines two classes of exception.

a) Errors that could potentially be caused by an attack by an adversary. When such an error occurs, no diagnostic information is returned, as such information might be useful for the adversary. Instead, it is assumed that an implementation logs the incident for later analysis.

b) Errors that may safely be assumed not to be caused by an adversary. In this case, a diagnostic message may be returned to the sender. It is recommended that implementations also log such incidents.

IETF RFC 5424 specifies classes of incidents. This Specification refers to two of those classes as follows.

a) Alert: Action shall be taken immediately. Errors that could potentially have been caused by an adversary.

b) Error: Exceptions assumed not to have been caused by an adversary.

In both cases, the association is either not established or it is aborted.

This Specification does not mandate any specific logging mechanism.

SECTION 2 – THE WRAPPER PROTOCOL

8 Wrapper protocol general concepts

8.1 Introduction

Section 2 specifies the general wrapper protocol providing security to other protocols. This clause considers the wrapper protocol in general. Clause 9 specifies the WrPDUs used to establish, terminate and abort associations. Clause 10 specifies the WrPDUs used during the data transfer phase. Clause 11 analyses the information flow. Clause 12 specifies a procedure for error checking.

The wrapper protocol requires that an association be established between two application entities before they go into the data transfer phase.

NOTE – The terms "association" and "application entity" are taken from the OSI area. However, these terms are here extended also to be applied to a non-OSI area.

The wrapper protocol makes use of parameterized data types to avoid specifications that relate to specific cryptographic algorithms keeping the protocol open to employ different cryptographic algorithms without changes to the basic protocol.

8.2 UTC time specification

The security of the wrapper protocol's shared key generation is based, among other things, on the presence of a trusted source of time on the network.

WrPDUs include information about the time of their creation. The time is given as a coordinated universal time (UTC)time.

This specification makes use of UTC time according to the following.

a) The year, months and date shall be written as YYYYMMDD, where YYYY is the year, MM is the number of the month and DD is the number of the day of the month.

b) The hour, minute and second and fraction of second shall be written as hhmmss,sss, where hh is the hour in the 24 h clock system, mm is the minute and the ss is the seconds. A comma (',') or full stop ('.') is used as the decimal separator.

c) The accuracy shall be either 1 h, 1 min, 1 s or a fraction of a second to any degree of accuracy.

d) The time shall be specified as a pure UTC time and a 'Z' shall be appended to the time. Other forms shall not be used.

e) There shall not be a 'T' between the data and the time specification.

The time value is given as a value of the **`GeneralizedTime`** ASN.1 data type.

8.3 Use of alternative cryptographic algorithms

Some WrPDUs allow for inclusion of alternative cryptographic algorithms. Such alternative cryptographic algorithms are intended for migrating cryptographic algorithms to more safe cryptographic algorithms.

The requirements for cryptographic mechanisms and algorithms acceptable for use in the wrapper protocol will be further defined in future versions of this Specification.

The cryptographic mechanisms and algorithms used by the end-users of the wrapper protocol shall comply with national regulations.

8.4 Establishment of shared keys

To encrypt a wrapped PrPDU or generate an ICV require the establishment of shared symmetric keys. Two types of symmetric keys are defined:

– content encryption keys, one for each direction if encryption of wrapped PrPDUs is required;

– ICV encryption keys, one for each direction, except if AEAD is used.

Different symmetric keys shall be used for the two directions of communication.

The classical way to establish shared symmetric encryption keys is to use a DH key agreement algorithm.

8.5 Sequence numbers

The wrapper protocol uses sequence number for WrPDUs transmitted during the data transfer phase, one sequence number per direction of communication. The sequence number shall take the value 0 for the first WrPDU sent after the establishment of an association and be incremented by one for each WrPDU sent. The purpose is:

– to allow detection of replay of WrPDUs caused by an error or by an adversary;

– to detect missing WrPDUs.

The sequence number is a signed integer with a value range from 0 to $2^{31} - 1$ (0 ... 2147483647) (max. 4 octets).

8.6 Use of invocation identification in the wrapper protocol

While protected protocols may have a requirement for pairing data requests and data responses, the same requirement may not be relevant for the wrapper protocol. However, there might be cases where the wrapper protocol needs to pair data requests and data responses, e.g., where non-repudiation is required for the protected protocol.

The invocation identification is held by an **invokeID** component having a syntax of a 6-octet character string where the first three characters are either REQ or RSP, depending on whether it a component of a data request or data response. The last three characters are numerals taking a value in the range 000 to 127, as given by the protected protocol.

8.7 Mapping to underlying services

The wrapper protocol maps directly on to the transmission control protocol (TCP) layer as specified by IETF RFC 793. The TCP port number 9877 as assigned by Internet Assigned Numbers Authority (IANA) to this Specification shall be used.

The service number assigned by IANA is x510.

8.8 Definition of protected protocols

A wrapped protocol is identified by an information object being an instance of the **WRAPPED-PROT** information object class. The **WRAPPED-PROT** information object class is equivalent to the **TYPE-IDENTIFIER** information object class defined by Rec. ITU-T X.681 | ISO/IEC 8824-2.

```
WRAPPED-PROT ::= TYPE-IDENTIFIER
```

This information object is used to bind the type of protected protocol identified by an object identifier to the abstract syntax of that protocol.

A protected protocol reference shall have the following syntax.

```
WrappedProt {WRAPPED-PROT:SupportedProtSet} ::= SEQUENCE {
  id    WRAPPED-PROT.&id({SupportedProtSet}),
  prot  WRAPPED-PROT.&Type({SupportedProtSet}{@id}),
  ... }
```

The **WrappedProt** parameterized data type has the following components:

a) the **id** component shall hold the object identifier assigned to the protected protocol in question;

b) The **prot** component shall hold an instance of the top level APDU being a choice of all the PrPDUs defined by the protected protocol (see Figure 4).

How the protected protocol is wrapped by the wrapper is further specified in Annex C.

8.9 Overview of wrapper protocol data units

The **WrapperPDU** data type represents the top APDU of the set of WrPDUs supported by the wrapper protocol.

```
WrapperPDU ::= CHOICE {
  handshakeReq       [0]  HandshakeReq,
  handshakeAcc       [1]  HandshakeAcc,
  handshakeWrpRej    [2]  HandshakeWrpRej,
  handshakeProRej    [3]  HandshakeProRej,
  handshakeSecAbort  [4]  HandshakeSecAbort,
  handshakeProAbort  [5]  HandshakeProAbort,
  dtSecAbort         [6]  DtSecAbort,
  applAbort          [7]  ApplAbort,
  releaseReq         [8]  ReleaseReq,
```

```
  releaseRsp         [9]  ReleaseRsp,
  dataTransferClient [10] DataTransferClient,
  dataTransferServer [11] DataTransferServer,
  ... }
```

9 Association management

9.1 Introduction to association management

An association is defined as a cooperative relationship between two application entities, which enables the communication of information and the coordination of their joint operation for an instance of communication.

The initiator of an association is called the client and the target of an association request is called the server.

9.2 Association handshake request

The **HandshakeReq** WrPDU is used by the client to initiate an association. It has the following syntax.

```
HandshakeReq ::= Signed{TbsHandshakeReq}

TbsHandshakeReq ::= SEQUENCE {
  version        Version DEFAULT {v1},
  prProt         WRAPPED-PROT.&id ({SupportedProtSet}),
  sigAlg         AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  altSigAlg  [0] AlgorithmIdentifier {{SupportedAltSignatureAlgorithms}} OPTIONAL,
  pkiPath        DER-PkiPath,
  assoID         AssoID,
  time           TimeStamp,
  keyEst         AlgorithmWithInvoke{{SupportedKeyEstablishmentAlgos}},
  altKeyEst  [1] AlgorithmWithInvoke{{SupportedAltKeyEstablishmentAlgos}} OPTIONAL,
  encr-mode      CHOICE {
    aead       [2] SEQUENCE SIZE (1..MAX) OF
      algo           AlgorithmIdentifier{{SupportedAeadAlgorithms}},
    non-aead   [3] SEQUENCE {
      encr       [0] SEQUENCE SIZE (1..MAX) OF
        algo           AlgorithmIdentifier{{SupportedSymmetricKeyAlgorithms}}
                         OPTIONAL,
      icvAlgID   [1] SEQUENCE SIZE (1..MAX) OF
        algo           AlgorithmIdentifier{{SupportedIcvAlgorithms}} },
    ... },
  attCert        DER-AttributeCertificate OPTIONAL,1
  applData   [4] WrappedProt{{SupportedProtSet}} OPTIONAL,
  ... }

Version ::= BIT STRING {
  v1 (0)  -- version 1
  }

DER-PkiPath ::= OCTET STRING
  (CONTAINING PkiPath ENCODED BY der)

DER-AttributeCertificate ::= OCTET STRING
  (CONTAINING AttributeCertificate ENCODED BY der)

der OBJECT IDENTIFIER ::=
  {joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1)}

AssoID ::= INTEGER (0.. 32767)

TimeStamp ::= GeneralizedTime

SupportedSignatureAlgorithms ALGORITHM ::= {...}

SupportedAltSignatureAlgorithms ALGORITHM ::= {...}

SupportedKeyEstablishmentAlgos ALGORITHM ::= {...}

SupportedAltKeyEstablishmentAlgos ALGORITHM ::= {...}
```

```
SupportedAeadAlgorithms ALGORITHM ::= {...}

SupportedSymmetricKeyAlgorithms ALGORITHM ::= {...}

SupportedIcvAlgorithms ALGORITHM ::= {...}
```

The **Signed** data type is defined in clause 6.5.2. As detailed in clause 8.3, a **Signed** data value may contain two digital signatures during a migration period. When a second digital signature is included, then the **altSigAlg** component shall also be present.

The **TbsHandshakeReq** data type has the following components.

a) The **version** component shall specify the version(s) of the wrapper protocol supported by the client. The **version** component is a bit-string that allows the client to set multiple bits if it supports multiple versions.

NOTE 1 – At the time of publication, only version 1 is defined.

b) The **prProt** component specifies the type of protocol to be protected. It shall hold the object identifier identifying that protocol.

c) The **sigAlg** component shall hold the digital signature algorithm to be used for the digital signature. If the **Signed** data value includes two digital signatures, then this component shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type.

The inclusion of the digital signature algorithm within the integrity-protected area serves two purposes. It allows the digital signature algorithm to be protected by the digital signature and by placing it near the start of the WrPDU makes it possible to generate the hash for digital signature verification during the first pass of the WrPDU.

NOTE 2 – This Specification does not mandate a specific set of digital signature algorithms. Reference specifications or implementers' agreements may replace the dots with a set of digital signature algorithms to be supported for a specific environment.

d) The **altSigAlg** component, when present, may be used for digital signature algorithm migration purposes (see clause 8.3). It shall be present if the **Signed** data value includes two digital signatures. Otherwise, it shall be absent. If present, it shall be used to generate the digital signature to be placed in the **altSignature** component of the **Signed** data type.

e) The **pkiPath** component shall hold the certification path necessary to verify the digital signature as specified in Rec. ITU-T X.509 | ISO/IEC 9594-8. The first element shall be a CA certificate issued by a trust anchor trusted by the relying party. The last element shall be the end-entity public-key certificate used to validate the digital signature. As a special case, the end-entity public-key certificate may be issued directly by the trust anchor. This component is encapsulated in an octet string being distinguished encoding rules (DER) encoded allowing other than basic encoding rules (BER) to be used for general encoding.

f) The **assoID** component shall uniquely identify the identity of an association within the context a client and server pair.

g) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

h) The **keyEst** component shall hold the key establishment algorithm used to establish shared symmetric keys to be used during data transfer. Two or four keys shall be generated as specified in item j).

i) The **altKeyEst** component, when present, may be used for key establishment algorithm migration.

j) The **encr-mode** component is a choice between two alternatives as follows.

– The **aead** alternative is used when the client wants to take advantage of the performance benefits of using an AEAD algorithm for both encryption and integrity protection using a single operation. When this alternative is taken, two symmetric keys shall be generated, one for each direction. The client may specify several AEAD algorithms. The client shall list the algorithms according to its preference by having the most preferred algorithm as the first one in the sequence-of. If a **HandshakeAcc** WrPDU is to be returned, the server shall take the same alternative and specify the first AEAD algorithm it supports of those suggested.

– The **non-aead** alternative is taken when the client does not suggest that encryption be used or the client for some reason wants added flexibility by using separate encryption and ICV specifications.

This alternative requires two symmetric keys to be generated for ICV purposes and an additional two symmetric keys to be generated if encryption is required.

This alternative has two components as follows.

i) In the **encr** component, when present, the client shall suggest a sequence-of one or more symmetric key algorithms listed according to its preference by having the most preferred algorithm as the first one in the sequence-of. If a **HandshakeAcc** WrPDU is to be returned, the server shall, if relevant, specify the first symmetric key algorithm it supports of those suggested.

ii) In the **icvAlgID** component, the client shall suggest a sequence-of one or more ICV algorithms listed according to its preference by having the most preferred algorithm as the first one in the sequence-of. If a **HandshakeAcc** WrPDU is to be returned, the server shall specify the first ICV algorithm it supports of those suggested.

k) The **attCert** component, when present, shall hold an attribute certificate providing access control information. This component is encapsulated in an octet string being DER encoded allowing other than BER to be used for general encoding.

l) The **applData** component, when required by the protected protocol, shall hold an instance of a PrPDU specifying some initialization information required by the protected protocol.

9.3 Association accept

The **HandshakeAcc** WrPDU shall be issued by the server, when accepting an association request.

The **HandshakeAcc** WrPDU has the following syntax.

```
HandshakeAcc ::= Signed{TbsHandshakeAcc}

TbsHandshakeAcc ::= SEQUENCE {
  version        Version DEFAULT {v1},
  sigSel         CHOICE {
    sigAlg         AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
    altSigAlg  [0] AlgorithmIdentifier{{SupportedAltSignatureAlgorithms}} },
  pkiPath        DER-PkiPath,
  assoID         AssoID,
  time           TimeStamp,
  keyEstSel      CHOICE {
    keyEst         AlgorithmWithInvoke{{SupportedKeyEstablishmentAlgos}},
    altKeyEst  [1] AlgorithmWithInvoke{{SupportedAltKeyEstablishmentAlgos}} },
  encr-mode      CHOICE {
    aead       [2] AlgorithmIdentifier{{SupportedAeaAlgorithms}},
    non-aead   [3] SEQUENCE {
      encr         [0] AlgorithmIdentifier{{SupportedSymmetricKeyAlgorithms}} OPTIONAL,
      icvAlgID     [1] AlgorithmIdentifier{{SupportedIcvAlgorithms}} },
    ... },
  attCert        DER-AttributeCertificate OPTIONAL,
  applData   [4] WrappedProt{{SupportedProtSet}} OPTIONAL,
  ... }
```

The **TbsHandshakeAcc** data type has the following components.

a) The **version** component shall specify exactly one version that is supported by the server. It shall be selected among those suggested in the corresponding **HandshakeReq**. The highest supported version of those suggested by the client should be selected.

b) The **sigSel** component has two alternatives:

– the **sigAlg** alternative shall be selected if the client did not include an alternative digital signature algorithm or if the client included an alternative algorithm, but the server does not support that alternative algorithm;

– the **altSigAlg** alternative shall be selected if the client has included an alternative digital signature algorithm and the server supports that algorithm.

The alternative taken determines which digital signature algorithm shall be used in future communications within the association.

The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the Signed data type shall be absent.

NOTE – If the **altSigAlg** alternative is taken, the parties in the communication now know that they both support the more secure digital signature algorithm, which may be utilized in future communications.

c) The **pkiPath** component shall hold the certification path used to verify the digital signature. This component shall be encapsulated in an octet string being DER encoded allowing other than BER to be used for general encoding.

d) The **assoID** component shall have the same value as specified for the corresponding component of the associated **HandshakeReq** WrPDU.

e) The **time** component shall hold the UTC generalized time at which of this component was created (see clause 8.2 for details).

f) The **keyEstSel** component has two alternatives:

– the **keyEst** alternative shall be selected if the client did not include an alternative key establishment or if the client included an alternative algorithm, but the server does not support that alternative algorithm;

– the **altKeyEst** alternative shall be selected if the client has included an alternative key establishment algorithm and the server supports that algorithm.

g) The **encr-mode** component is a choice between two alternatives. The server shall take the same alternative as the client did in the **HandshakeReq** WrPDU:

– if the client selected the **aead** alternative, the server shall select the first supported AE-algorithm of the sequence-of of the AEAD-algorithms suggested by the client;

– if the client selected the **non-aead** alternative, the server shall include the same components as included in the **HandshakeReq** WrPDU:

i) if the client included the **encr** component, the server shall select the first supported symmetric key algorithm of the sequence-of of those suggested by the client,

ii) the **icvAlgID** component shall select the first supported ICV algorithm of the sequence-of of those suggested by the client.

h) The **attCert** component, when present, shall hold an attribute certificate. This component shall be encapsulated in an octet string being DER encoded allowing other than BER to be used for general encoding.

i) The **applData** component, when required by the protected protocol, shall hold an instance of a PrPDU responding to initialization information provided by the client.

9.4 Association reject due to security issues

When the server detects an error or an unsupported component in the wrapper protocol part of a **HandshakeReq** WrPDU, a **HandshakeWrpRej** WrPDU shall be returned.

The **HandshakeWrpRej** WrPDU has the following syntax.

```
HandshakeWrpRej ::= Signed{TbsHandshakeWrpRej}

TbsHandshakeWrpRej ::= SEQUENCE {
  version         Version DEFAULT {v1},
  sigSel          CHOICE {
    sigAlg          AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
    altSigAlg  [0] AlgorithmIdentifier{{SupportedAltSignatureAlgorithms}} },
  assoID          AssoID,
  time            TimeStamp,
  pkiPath         DER-PkiPath,
  diag            WrpError OPTIONAL,
  ... }
```

The **altAlgorithmIdentifier** and the **altSignature** components of the **Signed** data value shall be absent.

The **TbsHandshakeWrpRej** data type has the following components.

a) The **version** component shall specify exactly one version that is supported by the server. It should be selected among those suggested in the corresponding **HandshakeReq** WrPDU. The highest supported version of those suggested should be selected. However, if the server does not support any of the suggested versions, it shall return one alternative value it does support.

b) The **sigSel** component has two alternatives:

– the **sigAlg** alternative shall be selected if the client did not include an alternative digital signature algorithm or if the client included an alternative algorithm, but the server does not support that alternative algorithm, but it does support the native algorithm;

– the **altSigAlg** alternative shall be selected if the client has included an alternative digital signature algorithm and the server supports that algorithm.

If the server does not support any of the algorithms, it shall instead add a supported algorithm in the **sigAlg** alternative.

The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

c) The **assoID** component shall have the same value as specified for the corresponding component of the associated **HandshakeReq** WrPDU.

d) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

e) The **pkiPath** component shall hold the certification path used to verify the digital signature.

f) The **diag** component, when present, shall hold a **WrpError** value with the appropriate diagnostic code. It shall be absent if an alert event was raised. Otherwise, it shall be present.

9.5 Association reject by the protected protocol

When an error is detected in the protected protocol parts of a **HandshakeReq** WrPDU, a **HandshakeProRej** WrPDU shall be returned.

The **HandshakeProRej** WrPDU has the following syntax.

```
HandshakeProRej ::= Signed{TbsHandshakeProRej}

TbsHandshakeProRej ::= SEQUENCE {
  sigSel          CHOICE {
    sigAlg          AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
    altSigAlg  [0] AlgorithmIdentifier{{SupportedSignatureAlgorithms}} },
  assoID          AssoID,
  time            TimeStamp,
  pkiPath         DER-PkiPath,
  applData        WrappedProt{{SupportedProtSet}},
  ... }
```

The **TbsHandshakeProRej** WrPDU has the following components.

a) The **sigSel** component has two alternatives:

– the **sigAlg** alternative shall be selected if the client did not include an alternative digital signature algorithm or if the client included an alternative algorithm, but the server does not support that alternative algorithm;

– the **altSigAlg** alternative shall be selected if the client has included an alternative digital signature algorithm and the server supports that algorithm.

The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

b) The **assoID** component shall have the same value as specified by the corresponding component of the associated **HandshakeReq** WrPDU.

c) The **time** component shall hold the UTC generalized time at which of this component was created (see clause 8.2 for details).

d) The **pkiPath** component shall hold the certification path used to verify the digital signature.

e) The **applData** component shall hold information from the protected protocol as to the reason for the rejection.

9.6 Handshake security abort

A **HandshakeSecAbort** WrPDU is issued by the client when it does not agree with the information supplied in the **HandshakeAcc** WrPDU.

The **HandshakeSecAbort** WrPDU has the following syntax.

```
HandshakeSecAbort ::= Signed{TbsHandshakeSecAbort}

TbsHandshakeSecAbort ::= SEQUENCE {
  version      Version DEFAULT {v1},
  sigAlg       AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  diag         WrpError OPTIONAL,
  ... }
```

The **TbsHandshakeSecAbort** data type has the following components.

a) The **version** component shall have the same bits set as in the original **HandshakeReq** WrPDU.

b) The **sigAlg** component shall have the same value as the one used in the **HandshakeAcc** WrPDU, if that was a valid value according to the client. Otherwise, it shall hold the native digital signature algorithm as specified in the **sigAlg** component in the original **HandshakeReq** WrPDU.

   The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

c) The **AssoID** component shall hold the same value as in the rejected **HandshakeAcc** WrPDU.

d) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

e) The **pkiPath** component shall hold the certification path used to verify the digital signature.

f) The **diag** component, when present, shall hold a value of the **WrpError** data type with the appropriate diagnostic code. It shall be absent if an alert event was raised. Otherwise, it shall be present.

9.7 Handshake abort by protected protocol

A **HandshakeProAbort** WrPDU is issued by a client when the protected protocol does not agree with the information supplied in the **HandshakeAcc** WrPDU.

The **HandshakeProAbort** WrPDU has the following syntax.

```
HandshakeProAbort ::= Signed{TbsHandshakeProAbort}

TbsHandshakeProAbort ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  applData     WrappedProt{{SupportedProtSet}},
  ... }
```

The **TbsHandshakeProAbort** data type has the following components.

a) The **sigAlg** component shall have the same value as the one used in the **HandshakeAcc** WrPDU.

   The selected digital signature algorithm shall be used to generate the digital signature to be placed in the signature component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

b) The **AssoID** component shall hold the same value as in the rejected **HandshakeAcc** WrPDU.

c) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

d) The **pkiPath** component shall hold the certification path used to verify the digital signature.

e) The **applData** component shall hold information from the protected protocol as to the reason for the abortion.

9.8 Data transfer security abort

A **DtSecAbort** WrPDU is issued when a security problem is encountered within the wrapper protocol part during the data transfer phase. Integrity and authentication are provided by a digital signature rather than an ICV.

The **DtSecAbort** WrPDU has the following syntax.

```
DtSecAbort ::= Signed{TbsDtSecAbort}

TbsDtSecAbort ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  seq          SequenceNumber,
  diag         WrpError OPTIONAL,
  ... }
```

The **TbsDtSecAbort** data type has the following components.

a) The **sigAlg** component shall hold the digital signature algorithm resulting from the algorithm selection (see item b) in clause 9.3).

   The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

b) The **assoID** component shall have the same value as specified for the corresponding component of the associated **HandshakeReq** WrPDU.

c) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

d) The **pkiPath** component shall hold the certification path used to verify the digital signature.

e) The **seq** component shall hold the sequence number that caused the exception.

f) The **diag** component, when present, shall hold a value of the **WrpError** data type with the appropriate diagnostic code. It shall be absent if an alert event was raised. Otherwise, it shall be present.

9.9 Abort by protected protocol

An **ApplAbort** WrPDU shall be issued when a protected protocol during the data transfer phase decides to abort an association.

The **ApplAbort** WrPDU has the following syntax.

```
ApplAbort ::= Signed{TbsApplAbort}

TbsApplAbort ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  seq          SequenceNumber,
  applData     WrappedProt{{SupportedProtSet}},
  ... }
```

The **TbsApplAbort** data type has the following components.

a) The **sigAlg** component shall hold the digital signature algorithm resulting from the algorithm selection (see item b) in clause 9.3).

   The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

b) The **assoID** component shall have the same value as specified for the corresponding component of the associated **HandshakeReq** WrPDU.

c) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

d) The **pkiPath** component shall hold the certification path used to validate the digital signature.

e) The **seq** component shall hold the sequence number of WrPDU that held the PrPDU that caused the exception.

f) The **applData** component shall hold information from the protected protocol as to the reason for the abortion.

9.10 Release request WrPDU

A **ReleaseReq** WrPDU shall be issued when one of the application entities requires termination of an existing association. If the client and the server almost simultaneously issue a **ReleaseReq** WrPDU, we have a release collision as discussed in clause 9.12.

The **ReleaseReq** WrPDU has the following syntax.

```
ReleaseReq ::= Signed{TbsReleaseReq}

TbsReleaseReq ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  ... }
```

A **ReleaseReq** PDU has the following components.

a) The **sigAlg** component shall hold the digital signature algorithm resulting from the algorithm selection (see item b) in clause 9.3).

The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

b) The **assoID** component shall have the same value as specified for the corresponding component of the associated **HandshakeReq** WrPDU.

c) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

d) The **pkiPath** component shall hold the certification path used to verify the digital signature.

9.11 Release response WrPDU

A **ReleaseRsp** WrPDU shall be issued as a reply to a **ReleaseReq** WrPDU.

The **ReleaseRsp** WrPDU has the following syntax.

```
ReleaseRsp ::= Signed{TbsReleaseRsp}

TbsReleaseRsp ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  ... }
```

A **ReleaseRsp** WrPDU has the following components.

a) The **sigAlg** component shall hold the digital signature algorithm resulting from the algorithm selection (see item b) in clause 9.3).

The selected digital signature algorithm shall be used to generate the digital signature to be placed in the **signature** component of the **Signed** data type. The **altSignature** component of the **Signed** data type shall be absent.

b) The **assoID** component shall have the same value as specified for the corresponding component of the associated **HandshakeReq** WrPDU.

c) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

d) The **pkiPath** component shall hold the certification path used to verify the digital signature.

9.12 Release collision

When the client and the server almost simultaneously issue a **ReleaseReq** WrPDU, the result is a release collision as illustrated in Figure 6.

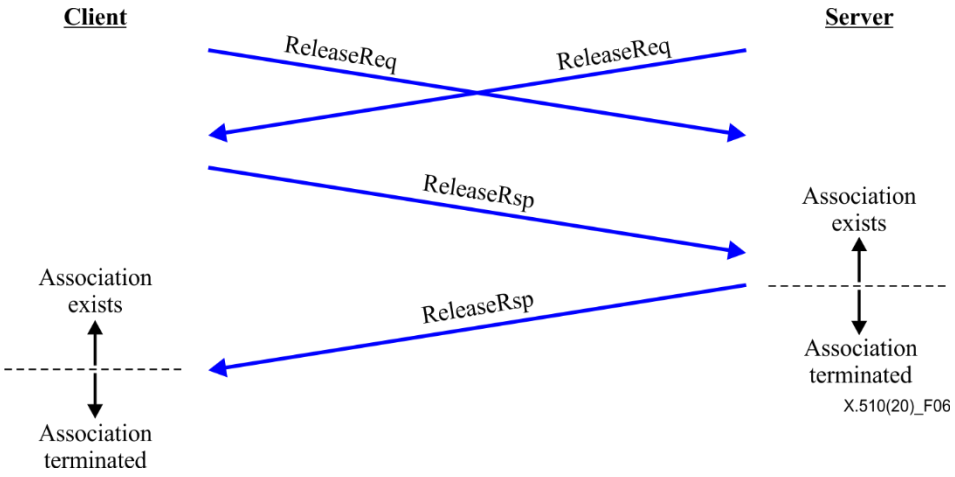

Figure 6 – Release collision

A collision is detected when both entities receive a **ReleaseReq** WrPDU instead of a **ReleaseRsp** WrPDU after having issued a **ReleaseReq** WrPDU. Having detected the collision situation:

1. the client shall issue a **ReleaseRsp** WrPDU;
2. the server shall wait for the **ReleaseRsp** WrPDU to arrive from the client;
3. when the server receives the **ReleaseRsp** WrPDU, it shall issue a **ReleaseRsp** WrPDU toward the client and shall consider the association as terminated;
4. when the client receives the **ReleaseRsp** WrPDU form the server, it shall consider the association as terminated.

10 Data transfer phase

10.1 Symmetric keys renewal

Symmetric keys used for encryption as well as for generation and verification of the ICV require renewal. This shall be done within the data transfer phase.

Only one side of a communication may initiate key renewal. Otherwise, the collision cases become quite complicated. Therefore, only the client may initiate key renewal. The wrapping protocol is therefore slightly different for data transfer initiated by the client and for data transfer initiated by the server.

10.2 Data transfer by the client

10.2.1 General

The **DataTransferClient** WrPDU is used by the client when transmitting data. It has the following syntax.

```
DataTransferClient ::= CHOICE {
  aead     [0] DataTransferClientAE,
  non-aead [1] DataTransferClientNEA,
  ... }
```

The **DataTransferClient** is a choice between the two alternative types of WrPDUs.

a) The **aead** alternative shall be selected if the **aead** alternative was taken for the **encr-mode** component of the **HandshakeReq** WrPDU. It is further discussed in clause 10.2.2.

b) The **non-aead** alternative shall be selected if the **non-aead** alternative was taken for the **encr-mode** component of the **HandshakeReq** WrPDU. It is further discussed in clause 10.2.3.

10.2.2 Client using authenticated encryption with associated data

The **DataTransferClientAE** WrPDU is expressed using the **AUTHEN-ENCRYPT** parameterized data type specified in clause 6.6.2.

```
DataTransferClientAE ::= AUTHEN-ENCRYPT{AadClientAE, WRAPPED-PROT.&Type}

AadClientAE ::= SEQUENCE {
  COMPONENTS OF    AadClient,
  encInvoke    [3] AlgoInvoke{{SupportedAeadAlgorithms}} OPTIONAL,
  ... }
```

The **DataTransferClientAE** WrPDU has the following parameters.

a) The **AadClientAE** is the part of the **DataTransferClientAE** that is not encrypted, but serves as associated data for the AEAD algorithm. It consists of the components of the **AaDClient** data type (see clause 10.2.4) together with the value of the dynamic parameters, if any, for the AEAD algorithm, as specified by the **encInvoke** component.

b) The **WRAPPED-PROT.&Type** is the PrPDU to be encrypted.

10.2.3 Client not using authenticated encryption with associated data

The **DataTransferClientNEA** WrPDU has the following syntax.

```
DataTransferClientNEA ::= ICV-Invoke{TbpDataTransferClient}

TbpDataTransferClient ::= SEQUENCE {
  COMPONENTS OF    AadClient,
  encEnvoke    [3] AlgoInvoke{{SupportedSymmetricKeyAlgorithms}} OPTIONAL,
  conf             CHOICE {
    clear        [4] WrappedProt{{SupportedProtSet}},
    protected    [5] ENCIPHERED{WRAPPED-PROT.&Type},
    ... },
  ... }
```

The **TbpDataTransferClient** data type has the following components.

a) The components of the **AadClient** data type are defined in clause 10.2.4.

b) The **encEnvoke** component, when present, shall hold the value of the dynamic parameters of the information object for the symmetric key algorithm in question. It shall be present if encryption is required and if that algorithm has dynamic parameters. Otherwise, this component shall be absent.

NOTE – It the symmetric key algorithm is of the advanced encryption standard-cipher block chaining (AES-CBC) type, the dynamic parameter is an initialization vector of 16 octets.

c) The **conf** components has two alternatives:

- the **clear** alternative shall be taken if confidentiality (encryption) is not required and shall then include the appropriate PrPDU of the protected protocol;
- the **protected** alternative shall be taken if confidentiality is required and shall then include an encrypted PrPDU of the protected protocol.

10.2.4 Client non-encrypted data

Client non-encrypted data in the form of the **AadClient** data type is included in a data transfer WrPDU as specified in clauses 10.2.2 and 10.2.3.

The **AadClient** data type has the following syntax.

```
AadClient ::= SEQUENCE {
  invokeID [0] InvokeID OPTIONAL,
  assoID       AssoID,
  time         TimeStamp,
  seq          SequenceNumber,
  keyEst   [2] AlgoInvoke{{SupportedKeyEstablishmentAlgos}} OPTIONAL }

InvokeID ::= OCTET STRING (SIZE (6))

SequenceNumber ::= INTEGER (0..2147483647)
```

The **AadClient** data type has the following components.

a) The **invokeID** component, when present, shall have a value determined by the protected protocol (see clause 8.6).

b) The **assoID** component shall take the value agreed for the association to which this WrPDU belongs.

c) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

d) The **seq** component shall hold a sequence number for the **DataTransferClient** WrPDU. The first data transfer WrPDU sent by the client within a specific association shall have sequence number '0' and increased by '1' for each new WrPDU to be sent. When reaching maximum, the sequence number shall wrap to '0'.

e) The **rekey** component shall be present when the client is required to refresh the shared keys.

The time maximum between key refreshments shall be configurable from a minimum of 15 min to a maximum of 24 h. The actual value is determined by local security policy.

The client shall not include the **rekey** component if it has not received a **DataTransferServer** WrPDU with the **changedKey** component set to **TRUE** for an outstanding **DataTransferClient** WrPDU with the **rekey** component included.

10.3 Data transfer by the server

10.3.1 General

The **DataTransferServer** WrPDU is used by the server when transmitting data. It has the following syntax.

```
DataTransferServer ::= CHOICE {
  aead     [0] DataTransferServerAE,
  non-aead [1] DataTransferServerNEA,
  ... }
```

The **DataTransferServer** is a choice between the two alternative types of WrPDUs:

a) the **aead** alternative shall be selected if the **aead** alternative was taken for the **encr-mode** component of the **HandshakeReq**;

b) the **non-aead** alternative shall be selected if the **non-aead** alternative was taken for the **encr-mode** component of the **HandshakeReq**.

10.3.2 Server using authenticated encryption with associated data

The **DataTransferServerAE** WrPDU is expressed using the **AUTHEN-ENCRYPT** parameterized data type specified in clause 6.6.2.

```
DataTransferServerAE ::= AUTHEN-ENCRYPT{AadServerAE, WRAPPED-PROT.&Type}

AadServerAE ::= SEQUENCE {
  COMPONENTS OF AadServer,
  encInvoke [3] AlgoInvoke{{SupportedAeadAlgorithms}} OPTIONAL,
  ... }
```

The **DataTransferServerAE** WrPDU has the following components.

a) The **AadServerAE** is the part of the **DataTransferServerAE** that is not encrypted, but serves as associated data for the AEAD algorithm. It consists of the components of the **AadServer** data type (see clause 10.3.4) together with the value of the dynamic parameters for the AEAD algorithm, if any, as specified by the **encInvoke** component.

b) The **WRAPPED-PROT.&Type** is the PrPDU to be encrypted.

10.3.3 Server not using authenticated encryption with associated data

The **DataTransferServerNEA** WrPDU has the following syntax.

```
DataTransferServerNEA ::= ICV-Invoke{TbpDataTransferServer}

TbpDataTransferServer ::= SEQUENCE {
  COMPONENTS OF     AadServer,
  encInvoke     [3] AlgoInvoke{{SupportedSymmetricKeyAlgorithms}} OPTIONAL,
```

```
  conf              CHOICE {
    clear         [4] WrappedProt{{SupportedProtSet}},
    protected     [5] ENCIPHERED{WRAPPED-PROT.&Type},
    ... },
  ... }
```

The **TbpDataTransferServer** data type has the following components.

a) The components of the **AadServer** data type as defined in clause 10.3.4.

b) The **encEnvoke** component, when present, shall hold the value of the dynamic parameters of the information object for the symmetric key algorithm in question. It shall be present if encryption is required and if the algorithm has dynamic parameters. Otherwise, this component shall be absent.

NOTE – It the symmetric key algorithm is of type AES-CBC, the dynamic parameter is an initialization vector of 16 octets.

c) The **conf** components has two alternatives:

– the **clear** alternative shall be taken if confidentiality (encryption) is not required and shall then include the appropriate PrPDU of the protected protocol;

– the **protected** alternative shall be taken if confidentiality is required and shall then include an encrypted PrPDU of the protected protocol.

10.3.4 Server non-encrypted data

Server non-encrypted data in the form of the **AadServer** data type is included in a data transfer WrPDU as specified in clauses 10.3.2 and 10.3.3.

The **AadServer** data type has the following syntax.

```
AadServer ::= SEQUENCE {
  invokeID   [0] InvokeID OPTIONAL,
  assoID         AssoID,
  time           TimeStamp,
  seq            SequenceNumber,
  reqRekey   [1] BOOLEAN DEFAULT FALSE,
  changedKey [2] BOOLEAN DEFAULT FALSE }
```

The **AadServer** data type has the following components.

a) The **invokeID** component, when present, shall have a value determined by the protected protocol (see clause 8.6).

b) The **assoID** component shall take the value for the association to which this WrPDU belongs.

c) The **time** component shall hold the UTC generalized time at which this component was created (see clause 8.2 for details).

d) The **seq** component shall hold a sequence number for the WrPDU. The WrPDU sent by the server within a specific association shall have sequence number '0' and increased by '1' for each new WrPDU to be sent. When reaching maximum, the sequence number shall wrap to '0'.

e) The **reqRekey** component shall be included with the value **TRUE** when the server wants the client to initiate a key renewal process. Otherwise, it shall have the value **FALSE** or be absent. This component shall not be included with the value **TRUE** if the server has received a **DataTransferClient** WrPDU with a **rekey** component without having sent a **DataTransferServer** WrPDU with a **changedKey** component confirming the latest **rekey** request.

f) The **changedKey** component shall be included with the value **TRUE** when the server has received a **DataTransferClient** WrPDU with a valid **rekey** component. The server shall use the old symmetric keys for the WrPDU holding this component. Subsequent **DataTransferServer** WrPDUs sent by the server shall be generated using the new keys. This component shall not be included with the value **TRUE** if there is not an outstanding **DataTransferClient** WrPDU with the **rekey** component present.

11 Information flow

11.1 Purpose and general model

This clause explores the flow of information between two application entities and verifies the control of that information flow.

Figure 7 depicts a general model describing a component going into a communication between two application entities. Figure 7 is not intended to reflect a possible implementation structure, but is used to reflect the relationship between the different aspects of an application entity.

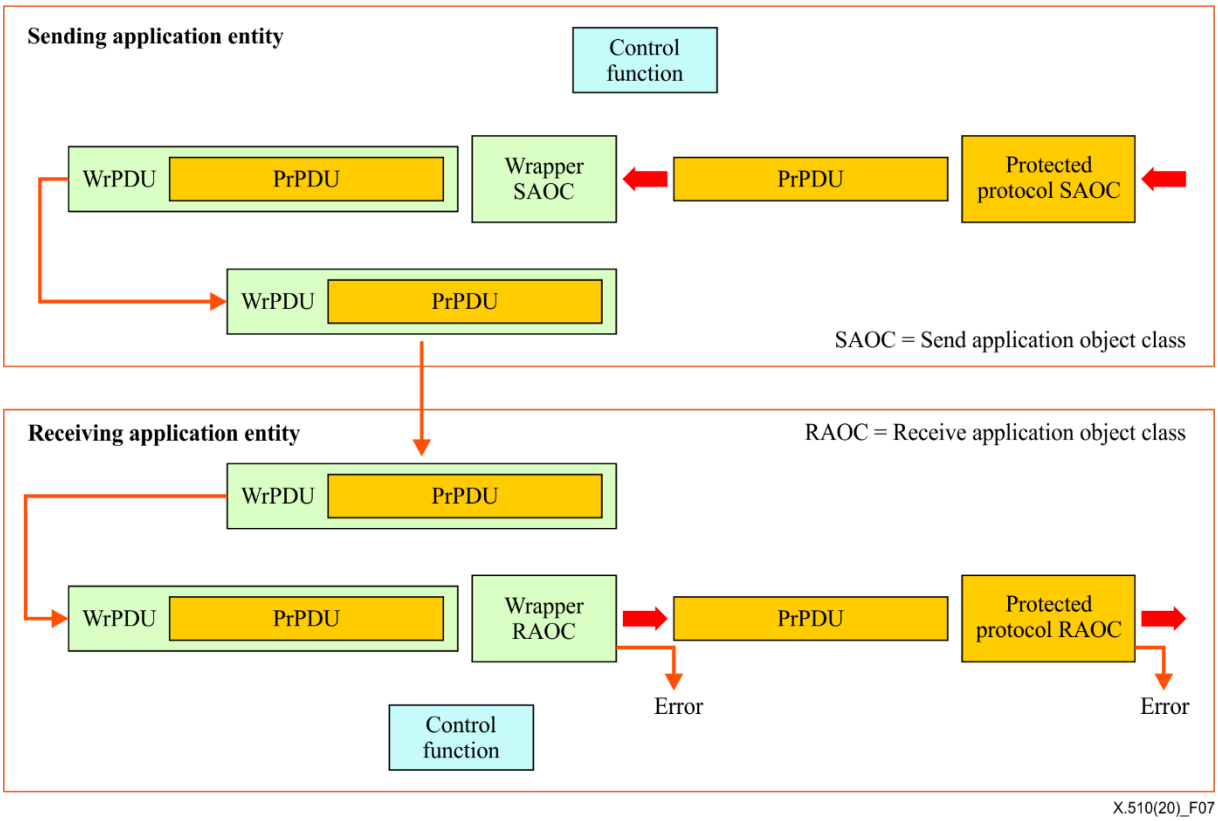

Figure 7 – Application entity structure and information flow

The concept of application entity has its origin in the Open Systems Interconnection (OSI) world, but it is here generalized to reflect the structure of a set of application functions making up an overall application specification for a specific purpose. This Specification therefore describes the structure of an application entity by borrowing some of the concepts specified in Rec. ITU-T X.207 | ISO/IEC 9545, although there are significant differences. It refers to the different elements as:

a) send application object classes (SAOCs) for controlling APDU generation;

b) receive application object classes (RAOCs) for validating received APDUs.

It takes an object-oriented view of the components, although a rather simple one.

Figure 7 also illustrates the information flow between two application entities, where the SAOCs are only shown in one application entity, while the RAOCs are shown in the other application entity. However, it should be realized that an application entity typically holds both types of objects.

As illustrated in Figure 7, an application entity in the context of this specification consists of two SAOCs and two RAOCs reflecting the wrapping protocol and the protected protocol. The control function included in Figure 7 models the overall coordination of the application entity. The implementation of the control function may be considered the overall application that includes the capabilities of the enclosing SAOCs and RAOCs.

Object classes defined in this way have subclasses for specific purposes. As an example, the wrapper SAOC has a subclass for each type of WrPDU to be generated. Each such subclass is described in terms of required input and generated output.

In the following, the SAOCs and RAOCs are described in more detail.

11.2 Protected protocol SAOC

There may be many different protected protocols to consider. The details of the protected protocol lie outside the scope of this Specification. The input to a protected protocol SAOC is the type of PrPDU to be generated and some additional information required by the PrPDU type.

Output from a protected protocol SAOC is a PrPDU to be submitted as input to the wrapper SAOC.

11.3 Wrapper SAOC

11.3.1 General

The input to the different subclasses of the wrapper SAOC is outlined in clauses 11.3.2 to 11.3.15.

The output from a subclass is a generated WrPDU to be transmitted.

11.3.2 Handshake request subclass

This subclass is invoked when the control function requests an association establishment. The control function provides the following input to the subclass:

a) the version information;

b) the object identifier for the protocol to be protected;

c) the cryptographic algorithms to be used and proposed;

d) whether encryption shall be supported;

e) if encryption shall be supported, whether AEAD shall be used or not;

f) the certification path to be included;

g) if relevant, an attribute certificate;

h) if relevant, a PrPDU to be included in the **`HandshakeReq`** WrPDU.

Based on this information a **`HandshakeReq`** WrPDU is generated ready for transmission.

11.3.3 Handshake accept subclass

This subclass is invoked when a received **`HandshakeReq`** WrPDU has been accepted. The control function provides the following input to the subclass:

a) the version to be used;

b) the cryptographic algorithms to be used;

c) encryption options;

d) the certification path to be included;

e) if relevant, an attribute certificate;

f) if relevant, a PrPDU to be included in the **`HandshakeAcc`** WrPDU.

11.3.4 Handshake security reject subclass

This subclass is invoked when a received **`HandshakeReq`** WrPDU has been rejected due to issues with the wrapper protocol elements. The control function provides the following input to the subclass:

a) version information according to clause 9.4;

b) the digital signature algorithm to be used;

c) the certification path to be included;

d) whether a diagnostic code shall be returned and if yes, with which diagnostic code.

11.3.5 Handshake reject by protected protocol subclass

This subclass is invoked when a received **`HandshakeReq`** WrPDU has been rejected due to issues detected by the protected protocol. This subclass is only relevant when the protected protocol has supplied initialization information. The control function provides the following input to the subclass:

a) the version to be used;

b) the digital signature algorithm to be used;

c) the certification path to be included;

d) a PrPDU from the protected protocol describing the issue.

11.3.6 Handshake security abort subclass

This subclass is invoked when a received **`HandshakeAcc`** WrPDU has been rejected due to issues with the wrapper protocol elements. The control function provides the following input to the subclass:

a) version information;

b) the digital signature algorithm to be used;

c) the certification path to be included;

d) whether a diagnostic code shall be returned and if yes, with which diagnostic code.

11.3.7 Handshake abort by protected protocol subclass

This subclass is invoked when a received **`HandshakeAcc`** WrPDU has been rejected due to issues detected by the protected protocol. This subclass is only relevant when the protected protocol has supplied initialization information. The control function provides the following input to the subclass:

a) the digital signature algorithm to be used;

b) the certification path to be included;

c) a PrPDU from the protected protocol describing the exception.

11.3.8 Data transfer security abort subclass

This subclass is invoked when due to security issues a received **`DataTransferClient`** WrPDU has been rejected by the server or when a received **`DataTransferServer`** WrPDU has been rejected by the client. The control function provides the following input to the subclass:

a) the digital signature algorithm to be used;

b) sequence number of faulty data transfer WrPDU;

c) the certification path to be included;

d) whether a diagnostic code shall be returned and if yes, with which diagnostic code.

11.3.9 Data transfer application abort subclass

This subclass is invoked when an exception condition is encountered in the protected protocol during data transfer. The control function provides the following input to the subclass:

a) the digital signature algorithm to be used;

b) sequence number of the wrapping WrPDU;

c) the certification path to be included;

d) a PrPDU from the protected protocol describing the exception.

11.3.10 Release request subclass

This subclass is invoked when the control function determines that the association should be terminated. The control function provides the following input to the subclass:

a) the digital signature algorithm to be used;

b) the association identifier (ID);

c) the certification path to be included.

Based on this information a **`ReleaseReq`** WrPDU is generated ready for transmission.

11.3.11 Release response subclass

This subclass is invoked when a **`ReleaseReq`** WrPDU was received. The control function provides the following input to the subclass:

a) the digital signature algorithm to be used;

b) the association ID;

c) the certification path to be included.

Based on this information a **ReleaseRsp** WrPDU is generated ready for transmission. If the entity acts as server for the association to be terminated and if a release collision is detected (see clause 9.12), the transmission of the WrPDU shall be deferred until a **ReleaseRsp** WrPDU is received from the client.

11.3.12 Client data transfer with authenticated encryption with associated data subclass

This subclass is invoked within the data transfer phase when AEAD is used. The control function provides the following input to the subclass:

a) the AEAD algorithm to be used;

b) the dynamic parameters associated with the AEAD algorithm;

c) the PrPDU to be protected;

d) if the protected protocol has a component that correspond to the **invokeID** component, the value of that component;

e) the association ID;

f) whether key renewal is to be initiated;

g) if key renewal is to be initiated, the key agreement algorithm to be used;

h) if key renewal is to be initiated, the dynamic parameters associated with the key agreement algorithm.

11.3.13 Client data transfer with integrity check value protection subclass

This subclass is invoked within the data transfer phase when protected by ICV. The control function provides the following input to the subclass:

a) whether encryption is used;

b) if encryption is used, the symmetric key algorithm;

c) if encryption is used, the dynamic parameters associated with the symmetric key algorithm;

d) the ICV algorithm to be used;

e) the dynamic parameters associated with the ICV algorithm;

c) the PrPDU to be protected;

d) whether key renewal is to be initiated;

e) if key renewal is to be initiated, the key agreement algorithm to be used;

f) if key renewal is to be initiated, the dynamic parameters associated with the key agreement algorithm;

g) if the protected protocol has a component that correspond to the **invokeID** component, the value of that component;

h) the association ID.

11.3.14 Server data transfer with authenticated encryption with associated data subclass

This subclass is invoked within the data transfer phase when AEAD is used. The control function provides the following input to the subclass:

a) the AEAD algorithm to be used;

b) the dynamic parameters associated with the AEAD algorithm;

c) the PrPDU to be protected;

d) if the protected protocol has a component that correspond to the **invokeID** component, the value of that component;

e) the association ID;

f) whether a reqRekey indication should be issued;

g) whether to respond to a key renewal by issuing a changedKey indication.

11.3.15 Client data transfer with integrity check value protection subclass

This subclass is invoked within the data transfer phase when a separate ICV is used, and if required, with a separate encryption of the embedded PrPDU. The control function provides the following input to the subclass:

a) whether encryption is used;

b) if encryption is used, the symmetric key algorithm;

c) if encryption is used, the dynamic parameters associated with the symmetric key algorithm;

d) the ICV algorithm to be used;

e) the dynamic parameters associated with the ICV algorithm;

f) the PrPDU to be protected;

g) if the protected protocol has a component that correspond to the **invokeID** component, the value of that component;

h) whether a reqRekey indication should be issued;

i) whether to respond to a key renewal by issuing a changedKey indication.

12 Wrapper error handling

12.1 General

This clause considers the type of errors that might occur within the WrPDU specific protocol elements.

12.2 Checking of a wrapper handshake request

12.2.1 General

If an exception condition is encountered in a **HandshakeReq** WrPDU, the server shall return a **HandshakeSecReject** WrPDU with the appropriate diagnostic code for the **diag** component, except when an alert event has been raised, in which case, the **diag** component shall be absent in the **HandshakeSecReject** WrPDU.

12.2.2 Digital signature checking

For security reasons, the digital signature shall be verified before any of the signed data is validated.

The **Signed** data value (see clause 8.3) is checked using the following procedure:

a) if neither of the digital signature algorithms specified in the **sigAlg** component nor the **altSigAlg** component (if present) are supported or accepted, then an **invalid-signatureAlgorithm** diagnostic code shall be returned;

b) if the **altSigAlg** component is absent or not supported, the **signature** components of the **Signed** data value shall be checked and if invalid, an invalid signature alert event shall be raised;

c) if the **altSigAlg** component is present and supported, then the **altSignature** component of the **Signed** data value shall be checked and if invalid, an invalid alternative signature alert event shall be raised.

12.2.3 Checking of the to-be-signed part

a) If none of the versions specified in the **HandshakeReq** WrPDU are supported by the server, an **unexpected-version** diagnostic code shall be returned.

b) If the protected protocol specified in the **prProt** component is not supported, a **protected-protocol-not-supported** diagnostic code shall be returned.

c) If the **pkiPath** component does not allow the verification of the attached public-key certificate, a public-key certificate verification failed alert event shall be raised.

d) If an association with an **assoID** and a **time** value equal to the one in the **HandshakeReq** WrPDU already exists between the two entities in question, a replay detected alert event shall be raised.

e) If an association with an **assoID** equal to the one in the **HandshakeReq** WrPDU already exists between the two entities in question, a **duplicate-assoID** diagnostic code shall be returned.

f) If the value in the **time** component is more than 5 min from the UTC time as observed locally, then an **invalid-time-value** diagnostic code shall be returned.

g) If none of the key establishment algorithms specified in the **keyEst** component or the **altKeyEst** component (if present) are supported, then a **key-estab-algorithm-not-supported** diagnostic code shall be returned.

h) If the **aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server does not support or does not want to support AEAD, an **encr-mode-aead-not-supported** diagnostic code shall be returned.

i) If the **aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server does not or does not want to support encryption, an **encryption-not-supported** diagnostic code shall be returned.

j) If the server does not support any of the AEAD algorithms proposed in the **encr-mode.aead.algo** component, then an **aead-algorithms-not-supported** diagnostic code shall be returned.

k) If the **non-aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server does not support or does not want to support encryption without authentication, an **aead-is-required** diagnostic code shall be returned.

l) If the **non-aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU with the **encr** component present and the server does not support or does not want to support encryption, then an **encryption-not-supported** diagnostic code shall be returned.

m) If the **non-aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU with the **encr** component absent and the server requires encryption, then an **encryption-required** diagnostic code shall be returned.

n) If the server does not support any of the symmetric key algorithms proposed in the **encr-mode.non-aead.encr.algo** component, then a **symmetricKey-algorithms-not-supported** diagnostic code shall be returned.

o) If the server does not support any of the ICV algorithms proposed in the **encr-mode.non-aead.icvAlgID.algo** component, then an **icv-algorithms-not-supported** diagnostic code shall be returned.

p) If the **attCert** component is present, the attribute certificate held by that component shall be checked for validity. If that check fails, the i**nvalid-attribute-certificate** diagnostic code shall be returned unless an alert event is to be raised where a diagnostic code shall not be returned.

12.3 Checking of a wrapper handshake accept

12.3.1 General

If an exception condition is encountered in a **HandshakeAcc** WrPDU, the client shall return a **HandshakeSecAbort** WrPDU with the appropriate diagnostic code, except when an alert event has been raised, in which case, the **diag** component shall not be included in the **HandshakeSecAbort** WrPDU.

12.3.2 Digital signature checking

For security reasons, the signature shall be verified before any of the signed data is validated.

The **Signed** data value (see clause 8.3) is checked using the following procedure:

a) if the **altSignature** components are included in the **Signed** data value, then an **alt-signature-not-allowed** diagnostic code shall be returned;

b) if the algorithm specified in the **sigAlg** component is different from both the **sigAlg** component and the **altSigAlg** component of the corresponding **HandshakeReq** WrPDU, then an invalid digital signature algorithm alert event shall be raised;

c) if the digital signature in the **signature** components of the **Signed** data value is not verified, an invalid signature alert event shall be raised.

12.3.3 Checking of the to-be-signed part

a) If a **version** component has bits set for more than one version, an **only-one-version** diagnostic code shall be returned.

b) If the version selected by the server was not included in the corresponding **HandshakeReq** WrPDU, an **unexpected-version** diagnostic code shall be returned.

c) If the **pkiPath** component does not allow the verification of the attached public-key certificate, a public-key certificate verification failed alert event shall be raised.

d) If the **assoID** does not identify an outstanding **HandshakeReq** WrPDU, discard the WrPDU and raise an invalid assoID error event.

e) If the value in the **time** component is more than 5 min from the UTC time as observed locally, then an **invalid-time-value** diagnostic code shall be returned.

f) If the **keyEst** alternative is taken for the **keyEstSel** component, and if the specified algorithm is different from the one specified in the **keyEst** component of the corresponding **HandshakeReq** WrPDU, then an **invalid-key-alt-estab-algorithm** diagnostic code shall be returned.

g) If the **altKeyEst** alternative is taken for the **keyEstSel** component, and if the specified algorithm is different from the one specified in the **altKeyEst** component of the corresponding **HandshakeReq** WrPDU or if the corresponding **HandshakeReq** WrPDU did not include that component, then an **invalid-key-alt-estab-algorithm** diagnostic code shall be returned.

h) If the **aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server in the **HandshakeAcc** WrPDU does not specify this alternative, then an **aead-is-required** diagnostic code shall be returned.

i) If the **aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server in the **encr-mode.aead** component of the **HandshakeAcc** WrPDU specifies an algorithm not included in the corresponding component of the **HandshakeReq** WrPDU, then an **invalid-aead-algorithm** diagnostic code shall be returned.

j) If the **non-aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server in the **HandshakeAcc** WrPDU does not specify this alternative, then an **aead-not-allowed** diagnostic code shall be returned.

k) If the **non-aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server in the **HandshakeAcc** WrPDU in the **encr-mode.non-aead.encr** component specifies an algorithm not included in the corresponding component of the **HandshakeReq** WrPDU, then an **invalid-symmetricKey-algorithm** diagnostic code shall be returned.

l) If the **non-aead** alternative of the **enc-mode** component is selected in a **HandshakeReq** WrPDU, but the server in the **HandshakeAcc** WrPDU in the **encr-mode.non-aead.icvAlgID** component specifies an algorithm that is not included in the corresponding component of the **HandshakeReq** WrPDU, then an **invalid-icv-algorithm** diagnostic code shall be returned.

m) If the **attCert** component is present, the attribute certificate held by that component shall be checked for validity. If that check fails, then an i**nvalid-attribute-certificate** diagnostic code shall be returned, unless an alert event is to be raised, in which case a diagnostic code shall not be returned.

12.4 Checking of data transfer WrPDUs

12.4.1 General

If an exception condition is encountered in the security parameters of a data transfer WrPDU, then a **DtSecAbort** WrPDU shall be returned with the appropriate diagnostic code in the **diag** component, except when an alert event has been raised, in which case, the **diag** component shall not be included in the **DtSecAbort** WrPDU.

12.4.2 Common checking for data transfer

12.4.2.1 Common checking for use of authenticate encryption with associated data

The following procedure shall be used when the **aead** alternative was taken for the **encr-mode** component of the **HandshakeReq** WrPDU.

a) If a **non-aead** alternative was taken for the data transfer, then an **aead-is-required** diagnostic code shall be returned.

b) If the authenticated encrypted WrPDU is not verified, an invalid AEAD alert event shall be raised.

c) If the **encInvoke** component of the **AadClientAE** / **AadServerAE** is required, then:

- if the **encInvoke** component is absent, a **dynamic-aead-algo-parms-required** diagnostic code shall be returned;
- if the AEAD dynamic parameters are invalid, an **invalid-dynamic-aead-algo-parms** diagnostic code shall be returned.

12.4.2.2 Common checking for non-use of authenticate encryption with associated data

The following procedure shall be used when the **non-aead** alternative was taken for the **encr-mode** component of the **HandshakeReq** WrPDU.

a) If the **aead** alternative was taken for the data transfer, then an **aead-not-allowed** diagnostic code shall be returned.

b) If the **encr-mode.non-aead.encr** component was present in the **HandshakeReq** WrPDU, then:

- if the data is not encrypted, return an **encryption-required** diagnostic code;
- if the symmetric key algorithm specifies dynamic parameters and values for these parameters are not included, return a **dynamic-symKey-algo-parms-required** diagnostic code;
- if the symmetric key algorithm specifies dynamic parameters and values for these parameters are invalid, return an **invalid-dynamic-symKey-algo-parms** diagnostic code;
- if the symmetric key algorithm does not specify dynamic parameters and values for parameters are included, return a **dynamic-symKey-algo-parms-not-required** diagnostic code;
- if decryption not possible, a decryption not possible alert event shall be raised.

c) If the **encr-mode.non-aead.encr** component was not present in the **HandshakeReq** WrPDU and data was encrypted, return an **encryption-not-supported** diagnostic code.

d) If the ICV algorithm specifies dynamic parameters and values for these parameters are not included, return a **dynamic-icv-algo-parms-required** diagnostic code.

e) If the ICV algorithm specifies dynamic parameters and values for these parameters are invalid, return an **invalid-dynamic-icv-algo-parms** diagnostic code.

f) If the ICV algorithm does not specify dynamic parameters and these parameters are included, return a **dynamic-icv-algo-parms-not-required** diagnostic code.

g) If the ICV is not verified, then an invalid ICV alert event shall be raised.

12.4.2.3 Common checking for AadClient and AadServer data types

The **AadClient** and -**AadServer** data types have some components in common. The components are considered in the following.

a) If the **invokeID** component is present, with a value starting with the first three characters “RSP” when no data WrPDU has been sent with the first three characters "REQ", an **unexpected-invokeID-received** diagnostic code shall be returned.

b) If the **assoID** component specifies an unknown association, the data WrPDU shall be discarded and an unknown assoID received alert event shall be raised.

c) If the value in the **time** component is more than 5 min from the UTC time as observed locally, then an **invalid-time-value** diagnostic code shall be returned.

d) If the value in the **seq** component is not '0' for the first data transfer WrPDU receive after association establishment or if the value is not '1' greater than the previous received data transfer WrPDU, then an invalid sequence number alert event shall be raised.

12.4.5 AadClient data value specific checking

When a server receives a **DataTransferClient** WrPDU, it shall perform the following specific check on the **AadClient** data value.

a) If the **rekey** component is present and the server has not yet replied with a **changedKey** indication to the **DataTransferClient** WrPDU with the **rekey** component, then a **rekey-out-of-sequence** diagnostic code shall be returned.

b) If the values for dynamic parameters of the key establishment algorithm are invalid, an **invalid-dynamic-keyEst-algo-parms** diagnostic code shall be returned.

12.4.6 AadServer data value specific checking

When a client receives a **DataTransferServer** WrPDU, it shall perform the following specific check on the **AadServer** data value.

a) If the **reqRekey** component is present with the value **TRUE**, when a **DataTransferClient** WrPDU with the **keyEst** component included and a **DataTransferServer** WrPDU with a **changedKey** component with the value **TRUE** has not been received, then the **reqRekey** component shall be ignored.

b) If the **changedKey** component is present with the value **TRUE**, when all previously sent **DataTransferClient** WrPDUs with the **keyEst** component included have already been confirmed with a **changedKey** with the value **TRUE**, then return a **changedKey-out-of-sequence** diagnostic code.

12.5 Wrapper diagnostic codes

The following diagnostic codes are defined for the wrapper protocol.

```
WrpError ::= ENUMERATED {
  protocol-error                          (0),
  invalid-signatureAlgorithm              (1),
  unexpected-version                      (2),
  protected-protocol-not-supported        (3),
  duplicate-assoID                        (4),
  invalid-time-value                      (5),
  key-estab-algorithm-not-supported       (6),
  encr-mode-aead-not-supported            (7),
  encryption-not-supported                (8),
  encryption-required                     (9),
  aead-algorithms-not-supported           (10),
  aead-is-required                        (11),
  symmetricKey-algorithms-not-supported   (12),
  icv-algorithms-not-supported            (13),
  invalid-attribute-certificate           (14),
  alt-signature-not-allowed               (15),
  only-one-version                        (16),
  invalid-key-estab-algorithm             (17),
  invalid-alt-key-estab-algorithm         (18),
  invalid-aead-algorithm                  (19),
  aead-not-allowed                        (20),
  invalid-symmetricKey-algorithm          (21),
  invalid-icv-algorithm                   (22),
  dynamic-aead-algo-parms-required        (23),
  invalid-dynamic-aead-algo-parms         (24),
  dynamic-aead-algo-parms-not-required    (25),
  dynamic-symKey-algo-parms-required      (26),
  invalid-dynamic-symKey-algo-parms       (27),
  dynamic-symKey-algo-parms-not-required  (28),
  dynamic-icv-algo-parms-required         (29),
  invalid-dynamic-icv-algo-parms          (30),
  dynamic-icv-algo-parms-not-required     (31),
  unexpected-invokeID-received            (32),
  rekey-out-of-sequence                   (33),
  invalid-dynamic-keyEst-algo-parms       (34),
  changedKey-out-of-sequence              (35),
  ... }
```

SECTION 3 – PROTECTED PROTOCOLS

13 Authorization and validation list management

13.1 General on authorization and validation management

13.1.1 Introduction

Authorization and validation management is concerned with how the authorizer maintains AVL information within the AVL entities the authorizer supports. This management is executed using the authorization and validation manage protocol (AVMP). The AVMP is designed to be protected by the wrapper protocol as specified in Section 2.

NOTE – For a definition of AVL and authorizer see clause 11 of Rec. ITU-T X.509 | ISO/IEC 9594-8.

It shall be the same trusted CA that has certified the end-entity public-key certificates for the authorizer and the AVL entities the authorizer serves. Each AVL entity shall be supplied with:

- its own end-entity public-key certificate and the corresponding private key to be used for digital signature generation;
- the end-entity public-key certificate for the authorizer;
- the CA certificate of the issuing CA for above end-entity public-key certificate.

When the AVMP is the protected protocol, the authorizer is the client of the association, while the AVL entity is the server.

13.1.2 Invocation identification

When an interaction consists of a request and a response, the two PrPDUs are tied together with an **InvokeID**, which is an integer that allows the response to be paired with the corresponding request.

The **InvokeID** shall be unique within the context of a client-server pair. When an interaction has completed, the **InvokeID** may be reused for a new interaction.

13.1.3 Exception conditions

There are three types of exception condition as follows.

a) Warning: this is an exception condition within a received request that is not considered severe enough to cause termination of an existing association. Exception information is provided as part of normal response. This exception type is only relevant for the data transfer phase.

b) Error: this is an exception condition severe enough to cause an existing association to be terminated, but it is determined that the error is not caused by an adversary. The association is terminated by issuing an abort PrPDU with the appropriate diagnostic code.

c) Alert: this is an exception condition that potentially has been caused by the action of an adversary. The association is terminated by issuing an **abort** PrPDU with a **noRreason** diagnostic code.

13.2 Defined protected protocol data unit types

The following PrPDU types are defined for the AVMP.

```
avlprot WRAPPED-PROT ::= {
                 Avlprot
  IDENTIFIED BY  id-avlprot }

Avlprot ::= CHOICE {
  initReq       [0]  InitializationRec,
  initAcc       [1]  InitializationAcc,
  initRej       [2]  InitializationRej,
  initAbt       [3]  InitializationAbort,
  certReq       [4]  CertReq,
  certRsp       [5]  CertRsp,
  addAvlReq     [6]  AddAvlReq,
  addAvlRsp     [7]  AddAvlRsp,
  replaceAvlReq [8]  ReplaceAvlReq,
  replaceAvlRsp [9]  ReplaceAvlRsp,
  deleteAvlReq  [10] DeleteAvlReq,
```

```
  deleteAvlRsp  [11] DeleteAvlRsp,
  rejectAVL     [12] RejectAVL,
  abort         [13] Abort,
  ... }
```

13.3 Authorization and validation management protocol initialization request

During association establishment, the version of AVMP is negotiated as part of the initialization. The **Initialization** PrPDU has the following syntax.

```
InitializationRec ::= SEQUENCE {
  version    Version,
  ... }
```

The **version** component shall hold the version of the AVMP. The syntax of the **Version** data type is a named bit string. The client may specify multiple versions by setting the appropriate bits, while the server shall specify only a single version among those suggested by the client.

The current version is version **v1**.

If the server supports at least one of the versions suggested by the client, it shall return **InitializationAcc** PrPDU. If the server does not support any of the versions suggested, it shall return an **InitializationRej** PrPDU.

The **InitializationReq** PrPDU shall be embedded in a **HandshakeReq** WrPDU (see clause 9.2).

13.4 Authorization and validation management protocol initialization accept

When accepting the initialization information in the **InitializationReq** PrPDU, the server shall return an **InitializationAcc** PrPDU. It has the following syntax.

```
InitializationAcc ::= SEQUENCE {
  version    Version,
  ... }
```

The **version** component shall specify a single version of those suggested by the client.

If the client does not accept the **InitializationAcc** PrPDU, it shall issue an **InitializationAbort** PrPDU.

The **InitializationAcc** PrPDU shall be embedded in a **HandshakeAcc** WrPDU (see clause 9.3).

13.5 Authorization and validation management protocol initialization reject

When rejecting the initialization information in the **InitializationReq** PrPDU, the server shall return an **InitializationRej** PrPDU. It has the following syntax.

```
InitializationRej ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersions     (0),
    ... },
  ... }
```

The **diag** component shall specify the reason for the rejection:

a) the server does not support any of versions proposed by the client.

The **InitializationRej** PrPDU shall be embedded in a **HandshakeProRej** WrPDU (see clause 9.5).

13.6 Authorization and validation management protocol initialization abort

When rejecting the initialization information in the **InitializationAcc** PrPDU, the client shall return an **InitializationAbort** PrPDU. It has the following syntax.

```
InitializationAbort ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion       (0),
    onlySingleVersionAllowed (1),
    ... },
  ... }
```

The **diag** component shall specify the reason for the abort:

a) the **unsupportedVersion** diagnostic code shall be selected if the server specifies a single version not included in those proposed by the client;

b) the **onlySingleVersionAllowed** diagnostic code shall be selected if the server returned multiple versions.

The **InitializationAbort** PrPDU shall be embedded in a **HandshakeProAbort** WrPDU (see clause 9.7).

13.7 Add authorization and validation list request

The authorizer uses the **addAvlReq** PrPDU to initiate the addition of an AVL to an AVL entity.

```
AddAvlReq ::= SEQUENCE {
  invokeID      InvokeID,
  certlist      CertAVL,
  ... }
```

The **AddAvlReq** PrPDU shall be embedded in a **DataTransferClient** WrPDU.

The **AddAvlReq** data type specifies the syntax of the actual content and has the following components:

a) the **invokeID** component shall identify the interaction consisting of the **AddAvlReq** and the corresponding **AddAvlRsp**;

b) The **certList** component shall hold the AVL to be added to the AVL entity.

The recipient AVL entity shall check the validity of the request:

a) by checking the validity of the signature on the received AVL and if invalid, raise an invalid AVL signature alert event and return a **noReason** error code in an **AbortAVL** PrPDU;

b) by checking whether all the AVL mandatory components are present and if not, return a **missingAvlComponent** error code;

c) by checking whether the **version** component on the received AVL validation list specifies a supported version and if not, return an **invalidAvlVersion** error code;

d) if the **serialNumber** component is present in the received AVL, then checking whether an AVL with the same value already exists and if so, return a **duplicateAVL** error code;

e) if the **serialNumber** component is absent in the received AVL, then checking whether an AVL with absent **serialNumber** component already exists and if so, return a **duplicateAVL** error code;

f) by checking whether the **constrained** component of the received AVL corresponds to the capabilities of the AVL entity and if not, return a **constrainedRequired** error code or a **nonConstrainedRequired** error code, as appropriate;

g) for each element of the **entries** component of the received AVL:

- by checking whether the **idType** component contains a valid alternative and if not, return a **protocolError** error code,
- if the **certIdentifier** alternative is chosen, then:
  - i) by checking whether the **certIdentifier** component contains a valid alternative and if not, return a **protocolError**,
  - ii) if the **entityGroup** alternative is taken and the **constrained** component has the value **TRUE**, return a **notAllowedForConstrainedAVLEntity** error code,
- by checking whether the **entryExtensions** component contains an unsupported critical extension and if so, return an **unsupportedCriticalEntryExtension** error code;

h) by checking whether the AVL contains an unsupported critical extension and if so, return an **unsupportedCriticalExtension** error code;

i) by checking whether the maximum number of AVLs has been exceeded by the new AVL and if so, return a **maxAVLsExceeded** error code.

NOTE – Maximum limit might be just a single AVL.

13.8 Add authorization and validation list response

The recipient AVL entity uses the **addAvlRsp** PrPDU to report the outcome of an add AVL.

```
AddAvlRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success [0]  AddAvlOK,
    failure [1]  AddAvlErr,
    ... },
  ... }

AddAvlOK ::= SEQUENCE {
  ok     NULL,
  ... }

AddAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }
```

The **AddAvlRsp** PrPDU shall be embedded in a **DataTransferServer** WrPDU.

The **AddAvlRsp** data type specifies the actual content and has the following components:

a) the **invokeID** component shall have the same value as in the corresponding **AddAvlReq** PrPDU;

b) the **result** component has the following two alternatives:

– the **success** alternative shall be taken if the addition of an AVL was performed successfully;

– the **failure** alternative shall be taken if the addition of an AVL failed – the **AVMP-error** data type is specified in clause 13.14.

13.9 Replace authorization and validation list request

The authorizer uses the **replaceAvlReq** PrPDU to initiate the replacement of an AVL at an AVL entity. It shall be used when changes to the AVL have occurred.

```
ReplaceAvlReq ::= SEQUENCE {
  invokeID     InvokeID,
  old          AvlSerialNumber OPTIONAL,
  new          CertAVL,
  ... }
```

The **ReplaceAvlReq** PrPDU shall be embedded in a **DataTransferClient** WrPDU.

The **ReplaceAvlReq** data type specifies the actual content and has the following components:

a) the **invokeID** component shall identify the interaction consisting of the **ReplaceAvlReq** and the corresponding **ReplaceAvlRsp**;

b) the **old** component, when present, shall hold the serial number of the old AVL and it shall be absent if the authorizer expects that an AVL with no sequence number exists;

c) the **new** component shall hold the replacement AVL.

The AVL entity shall verify the validity of the request by checking:

a) as specified in 13.7 items a) to h);

b) if the **old** component was present in the request, then check whether the **AvlSerialNumber** value specified in that component matches the **AvlSerialNumber** of a local authorization validation list and if not, return an **unknownAvl** error code;

c) if the **old** component was absent in the request, then check whether there is locally just a single AVL and that AVL is without the **serialNumber** component and if not, return an **unknownAvl** error code.

13.10 Replace authorization and validation list response

The AVL entity shall use the **replaceAvlRsp** PrPDU to report the outcome of an AVL replace request.

```
ReplaceAvlRsp ::= SEQUENCE {
```

```
  invokeID     InvokeID,
  result       CHOICE {
    success [0]  RepAvlOK,
    failure [1]  RepAvlErr,
    ... },
  ... }
RepAvlOK ::= SEQUENCE {
  ok     NULL,
  ... }

RepAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }
```

The **ReplaceAvlRsp** PrPDU shall be embedded in a **DataTransferServer** WrPDU.

The **ReplaceAvlRsp** PrPDU has the following components.

a) The **invokeID** component shall have the same value as in the corresponding **ReplaceAvlReq** PrPDU.

b) The **result** component has the following two alternatives:

   a) the **success** alternative shall be taken if the replacement of an AVL was performed successfully;

   b) the **failure** alternative shall be taken if the replacement of an AVL failed – the **AVMP-error** data type is specified in clause 13.14.

13.11 Delete authorization and validation list request

The authorizer uses the **deleteAvlReq** PrPDU to initiate deletion of an AVL at an AVL entity.

```
DeleteAvlReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  avl-Id        AVLSerialNumber OPTIONAL,
  ... }
```

The **DeleteAvlReq** PrPDU shall be embedded in a **DataTransferClient** WrPDU.

The **DeleteAvlReq** data type specifies the syntax of the actual content and has the following components:

a) the **invokeID** component shall identify the interaction consisting of the **ReplaceAvlReq** and the corresponding **ReplaceAvlRsp**;

b) the **avl-Id** component, when present, shall identify the AVL to be deleted.

The recipient AVL entity shall verify the validity of the request by:

a) if the **avl-id** component is present in the request, checking whether the **AlvSerialNumber** value specified in that component matches the **AvlSerialNumber** of a local AVL and if not, return an **unknownAVL** error code;

b) if the **avl-id** component is absent in the request, checking whether there locally is just a single AVL and that AVL is without the **serialNumber** component and if not, return an **unknownAVL** error code.

13.12 Delete authorization and validation list response

The recipient AVL entity uses the **deleteAvlRsp** content type to report the outcome of a delete AVL request.

```
DeleteAvlRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success [0]  DelAvlOK,
    failure [1]  DelAvlErr,
    ... },
  ... }

DelAvlOK ::= SEQUENCE {
  ok     NULL,
  ... }

DelAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }
```

The **deleteAvlRsp** PrPDU shall be embedded in a **DataTransferServer** WrPDU.

The **DeleteAvlRsp** PrPDU has the following components.

a) The **invokeID** component shall have the same value as in the corresponding **ReplaceAvlReq** PrPDU.

b) The **result** component has the following two alternatives:

– the **success** alternative shall be taken if the deletion of an AVL was performed successfully;

– the **failure** alternative shall be taken if the deletion of an AVL failed – the **AVMP-error** data type is specified in clause 13.14.

13.13 Authorization and validation list abort

The **AbortAvl** PrPDU is used by the authorizer to report problems with a response from the AVL entity.

```
AbortAVL ::= SEQUENCE {
  invokeID      InvokeID,
  reason        AVMP-error,
  ... }
```

The **AbortAvl** PrPDU has the following components:

a) the **invokeID** component shall have the same value as in the response being rejected;

b) the **AVMP-error** as specified in clause 13.14.

The **AbortAvl** PrPDU shall be embedded in an **ApplAbort** WrPDU.

13.14 Authorization and validation list error codes

A value of the **AVMP-error** data type is used by an AVL entity to report an error when processing a request from the authorizer. It is also used by an authorizer to reject a faulty response from an AVL entity.

```
AVMP-error ::= ENUMERATED {
  noReason                          (0),
  protocolError                     (1),
  duplicateAVL                      (2),
  missingAvlComponent               (3),
  invalidAvlVersion                 (4),
  notAllowedForConstrainedAVLEntity (5),
  constrainedRequired               (6),
  nonConstrainedRequired            (7),
  unsupportedCriticalEntryExtension (8),
  unsupportedCriticalExtension      (9),
  maxAVLsExceeded                   (10),
  unknownAVL                        (11),
  ... }
```

The following AVL error codes are defined:

a) the **noReason** value shall be selected when no other error code is applicable;

b) the **protocolError** value shall be selected when a protocol error is encountered;

c) the **duplicateAVL** value shall be selected when the authorizer attempts to add an already existing AVL to an end entity;

d) the **missingAvlComponent** value shall be selected when a received AVL is missing a mandatory component;

e) the **invalidAvlVersion** value shall be selected when an unsupported authorization validation list version is received;

f) the **notAllowedForConstrainedAVLEntity** shall be selected if a component is included that is not allowed by a constrained AVL entity;

g) the **constraintRequired** value shall be selected when the end entity requires an AVL with the **constraint** component set to **FALSE**;

h) the **nonConstraintRequired** value shall be selected when the end entity requires an AVL with the **constraint** component set to **TRUE**;

i) the **unsupportedCriticalEntryExtension** value shall be selected when a received AVL contains an unsupported critical entry extension;

j) the **unsupportedCriticalExtension** value shall be selected when a received AVL contains an unsupported critical extension;

k) the **maxAVLsExceeded** value shall be selected when the addition of an AVL would bring the number of AVLs beyond a locally determined value;

l) the **unknownAVL** value shall be selected when an end entity receives a content including a value of the **AvlSerialNumber** data type that did not match any local AVL.

14 Certification authority subscription protocol

14.1 Certification authority subscription introduction

The certification authority subscription protocol (CASP) is concerned with how the authorizer maintains AVL status information by subscribing to the necessary information from relevant CAs. It is only relevant for an authorizer supporting authorization validation lists for constraint AVL entities.

Before subscribing to maintenance information, the authorizer needs to know the exact public-key certification configuration for the end entities it supports. The following information is necessary to establish:

a) the end-entity public-key certificates for the AVL entities for which AVL support is to be provided;

b) for each AVL entity from a), the end-entity public-key certificates for the AVL entities to which communications are possible;

c) the CA-certificate and trust anchor information necessary to establish any necessary certification path.

This Specification does not give details on how an authorizer obtains this information. It could be by a local configuration or by extract of a centralized database.

The CASP comprises a set of exchange types as detailed in the following.

14.2 Defined protected protocol data unit types

```
casubprot WRAPPED-PROT ::= {
                  Casubprot
  IDENTIFIED BY  id-casubprot }

CasubProt ::= CHOICE {
  initReq             [0]  InitializationRec,
  initAcc             [1]  InitializationAcc,
  initRej             [2]  InitializationRej,
  initAbt             [3]  InitializationAbort,
  certSubscribeReq    [4]  CertSubscribeReq,
  certSubscribeRsp    [5]  CertSubscribeRsp,
  certUnsubscribeReq  [6]  CertUnsubscribeReq,
  certUnsubscribeRsp  [7]  CertUnsubscribeRsp,
  certReplaceReq      [8]  CertReplaceReq,
  certReplaceRsp      [9]  CertReplaceRsp,
  certUpdateReq       [10] CertUpdateReq,
  certUpdateRsp       [11] CertUpdateRsp,
  cAsubscribeAbort    [12] CAsubscribeAbort,
  ... }
```

14.3 Certification authority subscription protocol initialization request

During association establishment, the version of CASP is negotiated as part of the initialization. The **InitializationReq** PrPDU has the following syntax.

```
InitializationReq ::= SEQUENCE {
  version    Version,
  ... }
```

The **version** component shall hold the version of the CASP. The syntax of the version is a bit string. The client may specify multiple versions by setting the appropriate bits, while the server is required to specify only a single version among those suggested by the client.

The current version is version **v1**.

The **InitializationReq** PrPDU shall be embedded in a **HandshakeReq** WrPDU (see clause 9.2).

14.4 Certification authority subscription protocol initialization accept

When accepting the initialization information in the **InitializationReq** PrPDU, the server shall return an **InitializationAcc** PrPDU. It has the following syntax.

```
InitializationAcc ::= SEQUENCE {
  version    Version,
  ... }
```

The **version** component shall specify a single version of those suggested by the client.

If the client does not accept the **InitializationAcc** PrPDU, it shall issue an **InitializationAbort** PrPDU.

The **InitializationAcc** PrPDU shall be embedded in a **HandshakeAcc** WrPDU (see clause 9.3).

14.5 Certification authority subscription protocol initialization reject

When rejecting the initialization information in the **InitializationReq** PrPDU, the server shall return an **InitializationRej** PrPDU. It has the following syntax.

```
InitializationRej ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersions     (0),
    ... },
  ... }
```

The **diag** component signals the reason for the reject:

a) the server does not support any of versions proposed by the client.

The **InitializationRej** PrPDU shall be embedded in a **HandshakeProRej** WrPDU (see clause 9.5).

14.6 Certification authority subscription protocol initialization abort

When rejecting the initialization information in the **InitializationAcc** PrPDU, the client shall return an **InitializationAbort** PrPDU. It has the following syntax.

```
InitializationAbort ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion       (0),
    onlySingleVersionAllowed (1),
    ... },
  ... }
```

The **diag** component signals the reason for the abort:

a) the **unsupportedVersion** diagnostic code shall be selected if the server specifies a single version not included in those proposed by the client;

b) the **onlySingleVersionAllowed** diagnostic code shall be selected if the server returned multiple versions.

The **InitializationAbort** PrPDU shall be embedded in an **ApplAbort** WrPDU (see clause 9.8).

14.7 Public-key certificate subscription request

The authorizer uses the **CertSubscribeReq** PrPDU to request a specific CA to supply status information about public-key certificates issued by this CA relevant for the AVLs supported by the authorizer.

```
CertSubscribeReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs        SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }
```

The **CertSubscribeReq** PrPDU has the following components.

a) The **invokeID** component shall identify the interaction consisting of the **CertSubscribeReq a**nd the corresponding **CertSubscribeRsp**.

b) The **certs** component shall identify a list of end-entity public-key certificates, for which the authorizer requests information about status changes. It has the following subcomponents for each element:

– the **subject** subcomponent shall be the name of the entity for which the end-entity public-key certificate has been issued;

– the **serialNumber** subcomponent shall be the serial number for the end-entity public-key certificate in question.

The CA shall verify the validity of the request by checking:

a) each element of the **certs** component for validity, i.e., whether it identifies an end-entity public-key certificate issued by the CA – if not, an **unknownCert** status code shall be returned in the corresponding element of the response.

14.8 Public-key certificate subscription response

The CA uses the **CertSubscribeRsp** PrPDU to report the outcome of the subscription request.

```
CertSubscribeRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success       [0]  CertSubscribeOK,
    failure       [1]  CertSubscribeErr,
    ... },
  ... }

CertSubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok       [0] SEQUENCE {
    cert         Certificate,
    status       CertStatus,
    revokeReason CRLReason OPTIONAL,
    ... },
  not-ok   [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CASP-CertStatusCode ::= ENUMERATED {
  noReason                  (1),
  unknownCert               (2),
  ... }

CertStatus ::= ENUMERATED {
  good    (0),
  revoked (1),
  on-hold (2),
  expired (3),
  ... }

CertSubscribeErr ::= SEQUENCE {
  code       CASP-error,
  ... }
```

The **CertSubscribeRsp** data type specifies the actual content and has the following components.

a) The **invokeID** component shall have the same value as in the corresponding **CertSubscribeReq** PrPDU.

b) The **result** component has the following two alternatives:

– the **success** alternative shall be taken if the subscription was accepted for at least one public-key certificate – it shall then hold a value of the **CertSubscribeOK** data type;

– the **failure** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned – it shall then hold a value of the **CertSubscribeErr** data type – the **CASP-error** data type is specified in clause 14.16.

The **CertSubScribeOK** shall include an element for each public-key certificate specified in the request in the same order. Each element has two alternatives as follows.

a) The **ok** alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:
   - the **cert** component shall hold the public-key certificate for the requested subject;
   - the **status** component shall hold the status of the public-key certificate and shall take one the following values:
     i) the **good** value signals that the represented public-key certificate can be trusted,
     ii) the **revoked** value signals that the represented public-key certificate has been revoked and can no longer be trusted,
     iii) the **on-hold** value signals that the represented public-key certificate has been put on hold status and should not be trusted for the time being,
     iv) the **expired** value signals that the represented public-key certificate has expired and can no longer be trusted;
   - the **revokeReason** component may be present when the **status** component is set to **revoked**, and shall otherwise be absent. When present, it shall indicate the reason for the revocation as defined in clause 9.5.3.1.

b) The **not-ok** alternative shall be taken when a corresponding public-key certificate was not identified:
   - the **no-reason** status code shall be returned when no other code is applicable;
   - the **unknownCert** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

14.9 Public-key certificate un-subscription request

The authorizer uses the **CertUnsubscribeReq** PrPDU to request a specific CA to stop supplying status information about public-key certificates issued by that CA.

```
CertUnsubscribeReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs  SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }
```

The **CertUnsubscribeReq** PrPDU has the following components.

a) The **invokeID** component shall identify the interaction consisting of the **CertUnsubscribeReq a**nd the corresponding **CertUnsubscribeRsp**.

b) The **certs** component shall identify a list of public-key certificates for which the authorizer requests stop for information about status changes. It has the following subcomponents for each public-key certificate:
   - the **subject** subcomponent shall be the name of the entity to which the public-key certificate has been issued;
   - the **serialNumber** subcomponent shall be the serial number for the public-key certificate in question.

The CA shall verify the validity of the request by checking:

a) each element of the **certs** component for validity, i.e., whether it identifies a public-key certificate issued by the CA – if not, an **unknownCert** status code shall be returned in the corresponding element of the response.

14.10 Public-key certificate un-subscription response

The CA uses the **CertUnsubscribeRsp** PrPDU to report the outcome of the un-subscription request.

```
CertUnsubscribeRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success        [0]  CertUnsubscribeOK,
```

```
    failure       [1]  CertUnsubscribeErr,
    ... },
  ... }

CertUnsubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok       [0] SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  not-ok   [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CertUnsubscribeErr ::= SEQUENCE {
  code         CASP-error,
  ... }
```

The **CertSubscribeRsp** data type specifies the actual content and has the following components.

a) The **invokeID** component shall have the same value as in the corresponding **CertSubscribeReq** PrPDU.

b) The **result** component has the following two alternatives:

– the **success** alternative shall be taken if the subscription was accepted for at least one public-key certificate – it shall then hold a value of the **CertUnsubscribeOK** data type;

– the **failure** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned – it shall then hold a value of the **CertUnsubscribeErr** data type – the **CASP-error** data type is specified in clause 14.16.

The **CertUnsubScribeOK** includes an element for each public-key certificate specified in the request in the same order. Each element has two alternatives as follows.

a) The **ok** alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

– the **subject** component shall hold the name of the subject to which the public-key certificate had been issued;

– the **serialNumber** component shall hold the serial number for the public-key certificate.

b) The **not-ok** alternative shall be taken when a corresponding public-key certificate was not identified:

– the **no-reason** status code shall be returned when no other status code is applicable;

– the **unknownCert** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

The **error** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. The CASP-error data type is specified in clause 14.16.

14.11 Public-key certificate replacements request

The CA uses the **CertReplacementReq** PrPDU to submit replacement end-entity public-key certificates to the authorizer.

```
CertReplaceReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs         SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    old           CertificateSerialNumber,
    new           Certificate,
    ... },
  ... }
```

The **CertReplacementReq** PrPDU has the following components.

a) The **invokeID** component shall identify the interaction consisting of the **CertReplacementReq a**nd the corresponding **CertReplacementRsp**.

b) The **certs** component shall identify a list of public-key certificate replacements. It has the following subcomponents for each public-key certificate:

- the **old** subcomponent shall hold the identification of the public-key certificate to be replaced;
- the **new** subcomponent shall hold the replacement public-key certificate.

The authorizer shall verify the validity of the request by checking:

a) as specified in clause 7.3;

b) each element of the **certs** component for validity:

- whether the **old** subcomponent identifies a public-key certificate at the authorizer and if not, an **unknownCert** status code shall be returned in the corresponding element of the response;

c) each element of the **certs** component for validity, i.e., whether it identifies a public-key certificate issued by the CA – if not, an **unknownCert** status code shall be returned in the corresponding element of the response.

14.12 Public-key certificate replacement response

The authorizer uses the **CertReplaceRsp** PrPDU to report the outcome of the replacement request.

```
CertReplaceRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result        CHOICE {
    success       [0]  CertReplaceOK,
    failure       [1]  CertReplaceErr,
    ... },
  ... }

CertReplaceOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok        [0] SEQUENCE {
    issuer        Name,
    serialNumber  CertificateSerialNumber,
    ... },
  not-ok    [1] SEQUENCE {
    status        CASP-CertStatusCode,
    ... },
  ... }

CertReplaceErr ::= SEQUENCE {
  code          CASP-error,
  ... }
```

The **CertReplaceRsp** PrPDU has the following components.

a) The **invokeID** component shall have the same value as in the corresponding **CertSubscribeReq** PrPDU.

b) The **result** component has the following two alternatives:

- the **success** alternative shall be taken if the subscription was accepted for at least one public-key certificate – it shall then hold a value of the **CertReplaceOK** data type;
- the **failure** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned – it shall then hold a value of the **CertReplaceErr** data type – the CASP-error data type is specified in clause 14.16.

The **CertReplace** data type includes an element for each public-key certificate specified in the request in the same order. Each element has two alternatives as follows.

a) The **ok** alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

- the **subject** component shall hold the name of the subject to which the public-key certificate had been issued;
- the **serialNumber** component shall hold the serial number for the public-key certificate.

b) The **not-ok** alternative shall be taken when a corresponding public-key certificate was not identified:

- the **no-reason** status code shall be returned when no code is applicable;
- the **unknownCert** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

14.13 End-entity public-key certificate updates request

The CA uses the **CertUpdateReq** PrPDU to submit to the authorizer updated status information on public-key certificates.

```
CertUpdateReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs  SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    certStatus   CertStatus,
    ... },
  ... }
```

The **CertUpdateReq** PrPDU has the following components.

a) The **invokeID** component shall identify the interaction consisting of the **CertUpdateReq** and the corresponding **CertUpdateRsp**.

b) The **certs** component shall identify a list of updates to public-key certificates. It has the following subcomponents for each element:

- the **subject** subcomponent shall hold the identification of the end-entity public-key certificate to be replaced;
- the **serialNumber** subcomponent shall identify the end-entity public-key certificate or which new status information is available:
- the **certStatus** shall hold the updated status information for the end-entity public-key certificate in question.

The authorizer shall verify the validity of the request by checking:

a) each element of the **certs** component for validity by checking whether:

- the **subject** subcomponent identifies a new entity and if not, return an **unknownSubject** error code;
- the **serialNumber** subcomponent identifies a known public-key certificate and if not, return an **unknownCert** error code;
- the **certStatus** subcomponent has a valid value and if not, return an **unknownCertStatus** error code.

14.14 End-entity public-key certificate updates response

The authorizer shall use the **CertUpdateRsp** content type to report the outcome of the updates to status information on public-key certificates.

```
CertUpdateRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result        CHOICE {
    success       [0]  CertUpdateOK,
    failure       [1]  CertUpdateErr,
    ... },
  ... }

CertUpdateOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok         [0] SEQUENCE {
    subject       Name,
    serialNumber  CertificateSerialNumber,
    ... },
  not-ok     [1] SEQUENCE {
    status        CASP-CertStatusCode,
    ... },
  ... }

CertUpdateErr ::= SEQUENCE {
  code          CASP-error,
  ... }
```

The **CertUpdateRsp** PrPDU has the following components.

a) The **invokeID** component shall have the same value as in the corresponding **CertSubscribeReq** PrPDU.

b) The **result** component has the following two alternatives:

– the **success** alternative shall be taken if the subscription was accepted for at least one public-key certificate – it shall then hold a value of the **CertUpdateOK** data type;

– the **failure** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned – it shall then hold a value of the **CertUpdateErr** data type – the CASP-error data type is specified in clause 14.16.

The **CertUpdateOK** includes an element for each public-key certificate specified in the request in the same order. Each element has two alternatives as follows.

a) The **ok** alternative shall be taken when the update to the public-key certificate information was successfully processed. It has the following components:

– the **subject** component shall hold the name of the subject to which the public-key certificate had been issued;

– the **serialNumber** component shall hold the serial number for the public-key certificate.

b) The **not-ok** alternative shall be taken when a corresponding public-key certificate was not identified:

– the **no-reason** status code shall be returned when no code is applicable;

– the **unknownCert** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

14.15 Certification authority subscription abort

```
CAsubscribeAbort ::= SEQUENCE {
  invokeID     InvokeID,
  reason       CASP-error,
  ... }
```

The **CAsubscribeAbort** PDU type has the following components:

a) the **invokeID** component shall have the same value as in the response being rejected;

b) the **reason** component shall hold a **CASP-error** value as specified in clause 13.12.

14.16 Certification authority subscription error codes

```
CASP-error ::= ENUMERATED {
  noReason                    (0),
  unknownContentType          (1),
  unsupportedCASPversion      (2),
  missingContent              (3),
  missingContentComponent     (4),
  invalidContentComponent     (5),
  sequenceError               (6),
  unknownCertStatus           (7),
  ... }
```

A value of the **CASP-error** data type indicates the result of an issued request:

a) the **noReason** value shall be selected when no other error code is applicable;

b) the **unknownContentType** value shall be selected if the content type is not known by the receiver;

c) the **unsupportedCASPversion** value shall be selected if a request or response specified a CASP version that is not supported;

d) the **missingContent** value shall be selected when the request or response did not include content;

f) the **missingContentComponent** value shall be selected when a request or response does not include a mandatory component;

e) the **invalidContentComponent** value shall be selected when an unexpected component was included in a request or response;

f) the **sequenceError** value shall be selected by when:

- an authorizer or a CA receives a request for the first time that did not have the **sequence** component set to 1;
- an authorizer or a CA receives a request that did not have the **sequence** component set to one higher than for a previous request content in the same direction;
- an authorizer or a CA receives a response content with a **sequence** component value different from that in the corresponding request content.

15 Trust broker protocol

15.1 Introduction

A trust broker is an entity, possibly a commercial entity, that maintains reliability of one or more CAs and makes that information available to requesting relying parties. The trust broker protocol allows the relying party to query the trust broker about the trustworthiness of the public-key certificate issued to the subject it is communicating with. The relying party may send the public certificate of either the issuing CA or the subject and receive information about the trustworthiness of the subject's certificate.

There are two issues involved in determining the trustworthiness of a public-key certificate:

- whether the public-key certificate is still valid;
- the level of trustworthiness of the issuing CA.

The relying party may use a local client to answer the first point, in which case the trust broker will only answer the second. Alternatively, the relying party may ask the trust broker to answer both points.

The relying party requesting information shall act as the client of the association, while the trust broker acts the server.

15.2 Defined protected protocol data unit types

The following PDU types are defined for the trust broker protocol.

```
tbprot WRAPPED-PROT ::= {
                  TBprot
  IDENTIFIED BY  id-tbprot }

TBprot ::= CHOICE {

  caCert      [0] PKCertIdentifier,
  subjectCert [1] PKCertIdentifier,
  tbresponse  [2] CHOICE {
    success [0]  TBOK,
    failure [1]  TBerror,
    ... },
  ... }
```

15.3 Trust broker protocol initialization request

During association establishment, the version of the trust broker protocol is negotiated as part of the initialization. The **InitializationReq** PrPDU has the following syntax.

```
InitializationReq ::= SEQUENCE {
  version    Version,
  ... }
```

The **version** component shall hold the version of the trust broker protocol. The syntax of the version is a bit string. The client may specify multiple versions by setting the appropriate bits, while the server is required to specify only a single version among those suggested by the client.

The current version is version **v1**.

15.4 Trust broker protocol initialization accept

When accepting the initialization information in the **InitializationReq** PrPDU, the server shall return an **InitializationAcc** PrPDU. It has the following syntax.

```
InitializationAcc ::= SEQUENCE {
  version    Version,
  ... }
```

The **version** component shall specify a single version of those suggested by the client.

If the client does not accept the **InitializationAcc** PrPDU, it shall issue an **InitializationAbort** PrPDU.

This PrPDU shall be embedded in a **HandshakeAcc** WrPDU (see clause 9.3).

15.5 Trust broker protocol initialization reject

When rejecting the initialization information in the **InitializationReq** PrPDU, the server shall return an **InitializationRej** PrPDU. It has the following syntax.

```
InitializationRej ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersions     (0),
    ... },
  ... }
```

The **diag** component signals the reason for the reject:

a) the server does not support any of the versions proposed by the client.

This PrPDU shall be embedded in a **HandshakeProRej** WrPDU (see clause 9.5).

15.6 Trust broker protocol initialization abort

When rejecting the initialization information in the **InitializationAcc** PrPDU, the client shall return an **InitializationAbort** PrPDU. It has the following syntax.

```
InitializationAbort ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion       (0),
    onlySingleVersionAllowed (1),
    ... },
  ... }
```

The **diag** component signals the reason for the abort:

a) the **unsupportedVersion** diagnostic code shall be selected if the server specifies a single version not included in those proposed by the client;

b) the **onlySingleVersionAllowed** diagnostic code shall be selected if the server returned multiple versions.

This PrPDU shall be embedded in an **ApplAbort** WrPDU (see clause 9.8).

15.7 Trust broker request syntax

The syntax of format of a trust broker request is as follows.

```
TBrequest ::= CHOICE {
  caCert      [0] PKCertIdentifier,
  subjectCert [1] PKCertIdentifier,
  ... }
```

If the relying party sends the certificate of the CA, then only the second point in clause 15.1 is to be answered. The TB does not know which of the CA's subjects the relying party is communicating with, and therefore can only return information that relates to all certificates issued by this CA. This method protects the privacy of the relying party by not divulging the subjects that the relying party communicates with. The TB shall check that the CA's certificate is still valid and then return information about the trustworthiness of all the subject certificates issued by this CA.

If the relying party sends the certificate of the subject, then the TB shall check that the subject's certificate is still valid and then check that the CA's certificate is still valid and return information about the trustworthiness of all the subject certificates issued by this CA.

15.8 Trust broker response syntax

```
TBresponse ::= CHOICE {
  success [0]  TBOK,
  failure [1]  TBerror,
  ... }

TBOK ::= SEQUENCE {
  levelOfAssurance  [0]  INTEGER (0..100),
  confidenceLevel   [1]  INTEGER (0..100),
  validationTime    [2]  UTCTime,
  info                   UTF8String  OPTIONAL,
  ... }
```

If the trust broker successfully validates the presented public-key certificate, it returns a value of the TBOK data type. The TBOK data type has the following components.

a) The **levelOfAssurance** component shall indicate the level of assurance that the relying party can have in the subject's public-key certificate, as determined by the responding trust broker. This assurance is asserted by the trust broker as a result of its careful analysis of many factors. Different trust brokers may therefore return different levels of assurance values for the same presented public-key certificate due to their different validation procedures. The level of assurance value will be in the range 0 to 100, where 0 indicates zero assurance and 100 indicates full assurance.

b) The **confidenceLevel** component shall indicate the confidence that the trust broker has in the level of assurance value that it has returned, where 0 indicates zero confidence and 100 indicates full confidence.

c) The **validationTime** component shall hold the date and time at which the trust broker last checked the revocation information for the presented certificate.

d) The **info** component, when present, shall hold additional information to be presented to the requestor.

15.9 Trust broker error information

If the TB fails to validate the presented certificate, it returns an error code and diagnostic string. Note that if the TB validates the certificate but does not trust it or the issuer, it should return TBOK with a loss of alignment (LoA) of zero.

```
TBerror ::= SEQUENCE {
  code        ENUMERATED {
    caCertInvalid        (1),
    unknownCert          (2),
    unknownCertStatus    (3),
    subjectCertRevoked   (4),
    incorrectCert        (5),
    contractExpired      (6),
    pathValidationFailed (7),
    timeOut              (8),
    other                (99),
    ... },
  diagnostic  UTF8String OPTIONAL,
  ... }
```

The error code can be one of the following:

a) **caCertInvalid** – either the presented CA certificate is not valid, or the certificate of a CA superior to the presented subject certificate is not valid;

b) **unknownCert** – the certificate submitted to the TB by the relying party is unknown and cannot be validated;

c) **unknownCertStatus** – the TB cannot determine the status of the (known) certificate submitted by the relying party;

d) **subjectCertRevoked** – the presented subject certificate has been revoked;

e) **incorrectCert** – the presented certificate is incorrectly formatted;

f) **contractExpired** – the relying party's contract with the TB has expired and should be renewed before service can be restored;

g) **pathValidationFailed** – one or more of the CA certificates in the chain from the subject's certificate to the root CA could not be validated – another attempt at a later time may succeed;

h) **timeOut** – the presented certificate could not be validated because one or more services timed out – retrying again later may rectify this error;

i) **other** – the presented certificate could not be validated for some reason other than one of the above – if this error code is used, then it is mandatory for the TB to complete the diagnostic string.

The diagnostic string is an optional parameter that the TB may wish to be recorded in the client logs and displayed to the relying party. It is mandatory if the error code "other" is returned.

Annex A

Crypto Tools in ASN.1

(This annex forms an integral part of this Recommendation | International Standard.)

This annex includes all the ASN.1 type, value and information object definitions for cryptographic specifications and constitutes a formal ASN.1 module.

```
CryptoTools {joint-iso-itu-t ds(5) module(1) cryptoTools(42) 9}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

/*
Last component of object identifiers for X.510 modules

42 - CryptoTools
43 - Wrapper
44 - AVL-management
45 - CaSubscription
46 - TrustBroker
47 - ProtProtocols
48 - GenAlgo
*/

-- EXPORTS All

IMPORTS

  AlgoInvoke{}, ALGORITHM, AlgorithmIdentifier{}, AlgorithmWithInvoke{}
    FROM PKI-Stub
      {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9} WITH SUCCESSORS

  id-algo-mca
    FROM GenAlgo
      {joint-iso-itu-t ds(5) module(1) genAlgo(48) 9} WITH SUCCESSORS ;


multipleSignaturesAlgo ALGORITHM ::= {
  PARMS         MultipleSignaturesAlgo
  IDENTIFIED BY id-algo-multipleSignaturesAlgo }

MultipleSignaturesAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedSignatureAlgorithms}}

SupportedSignatureAlgorithms ALGORITHM ::= {...}

multipleSymmetricKeyAlgo ALGORITHM ::= {
  PARMS         MultipleSymmetricKeyAlgo
  IDENTIFIED BY id-algo-multipleSymmetricKeyAlgo }

MultipleSymmetricKeyAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedSymmetricKeyAlgorithms}}

SupportedSymmetricKeyAlgorithms ALGORITHM ::= {...}

multiplePublicKeyAlgo ALGORITHM ::= {
  PARMS         MultiplePublicKeyAlgo
  IDENTIFIED BY id-algo-multiplePublicKeyAlgo }

MultiplePublicKeyAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedPublicKeyAlgorithms}}

SupportedPublicKeyAlgorithms ALGORITHM ::= {...}

multipleHashAlgo ALGORITHM ::= {
  PARMS         MultipleHashAlgo
  IDENTIFIED BY id-algo-multipleHashAlgo }
```

```
MultipleHashAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedHashAlgorithms}}

SupportedHashAlgorithms ALGORITHM ::= {...}

multipleAuthenEncryptAlgo ALGORITHM ::= {
  PARMS         MultipleAuthenEncryptAlgo
  IDENTIFIED BY id-algo-multipleAuthenEncryptAlgo }

MultipleAuthenEncryptAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo       AlgorithmIdentifier{{SupportedAuthenEncryptAlgorithms}}

SupportedAuthenEncryptAlgorithms ALGORITHM ::= {...}

multipleIcvAlgo ALGORITHM ::= {
  PARMS         MultipleIcvAlgo
  IDENTIFIED BY id-algo-multipleIcvAlgo }

MultipleIcvAlgo ::= SEQUENCE SIZE (1..MAX) OF
  algo  AlgorithmIdentifier{{SupportedIcvAlgorithms}}

SupportedIcvAlgorithms ALGORITHM ::= {...}


-- Auxiliary data types

MULTY-SIGNED{ToBeSigned} ::= SEQUENCE {
  toBeSigned  ToBeSigned,
  algorithm   ALGORITHM.&id({multipleSignaturesAlgo}),
  parmeters     SEQUENCE SIZE (1..MAX) OF
    sign          SEQUENCE {
      algo          AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
      signature     BIT STRING,
      ... },
  ... }

Signed{ToBeSigned} ::= SEQUENCE {
  toBeSigned   ToBeSigned,
  signature    BIT STRING,
  altSignature BIT STRING OPTIONAL,
  ... }

ICV-Total{ToBeProtected} ::= SEQUENCE {
  toBeProtected               ToBeProtected,
  algorithmIdentifier         AlgorithmWithInvoke{{SupportedIcvAlgorithms}},
  icv                         BIT STRING,
  altAlgorithmIdentifier  [0] AlgorithmWithInvoke{{SupportedIcvAlgorithms}} OPTIONAL,
  altIcv                  [1] BIT STRING OPTIONAL,
  ... }
   (WITH COMPONENTS {..., altAlgorithmIdentifier PRESENT, altIcv PRESENT } |
    WITH COMPONENTS {..., altAlgorithmIdentifier ABSENT,  altIcv ABSENT } )

ICV-Invoke{ToBeProtected} ::= SEQUENCE {
  toBeProtected      ToBeProtected,
  dynParms       [0] AlgoInvoke{{SupportedIcvAlgorithms}} OPTIONAL,
  icv                BIT STRING,
  ... }

ENCIPHERED{ToBeEnciphered} ::= OCTET STRING (CONSTRAINED BY {
   -- shall be the result of applying an encipherment procedure
   -- to the BER-encoded octets of a value of -- ToBeEnciphered } )

AUTHEN-ENCRYPT{ToBeAuth, ToBeEnciphered} ::= SEQUENCE {
  aad  [0] ToBeAuth OPTIONAL,
  encr [1] ToBeEnciphered,
  ... }

-- Algorithms
```

```
id-algo-multipleSignaturesAlgo      OBJECT IDENTIFIER ::= {id-algo-mca 1}
id-algo-multipleSymmetricKeyAlgo    OBJECT IDENTIFIER ::= {id-algo-mca 2}
id-algo-multiplePublicKeyAlgo       OBJECT IDENTIFIER ::= {id-algo-mca 3}
id-algo-multipleHashAlgo            OBJECT IDENTIFIER ::= {id-algo-mca 4}
id-algo-multipleAuthenEncryptAlgo   OBJECT IDENTIFIER ::= {id-algo-mca 5}
id-algo-multipleIcvAlgo             OBJECT IDENTIFIER ::= {id-algo-mca 6}

END -- CryptoTools
```

Annex B

Wrapper protocol in ASN.1

(This annex forms an integral part of this Recommendation | International Standard.)

This annex includes all the ASN.1 type, value and information object definitions for the wrapper protocol in the form of the ASN.1 module **Wrapper**. This ASN.1 module is the formal specification of the wrapper protocol.

NOTE – This module was called **PkiPMIWrapper** in Rec. ITU-T X.509 (2016) | ISO/IEC 9594-8 :2017. The name is changed to indicate that the module is also relevant outside the strict PKI/PMI area.

```
Wrapper {joint-iso-itu-t ds(5) module(1) wrapper(43) 9}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All

IMPORTS

  AlgoInvoke{}, ALGORITHM, AlgorithmIdentifier{}, AlgorithmWithInvoke{},
AttributeCertificate, id-wrprot, PkiPath
    FROM PKI-Stub
      {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9}

  SupportedProtSet
    FROM ProtProtocols
      {joint-iso-itu-t ds(5) module(1) protProtocols(47) 9} WITH SUCCESSORS

  -- from Rec. ITU-T X.510 | ISO/IEC 9594-11

  AUTHEN-ENCRYPT{}, ENCIPHERED{}, ICV-Invoke{}, Signed{}
    FROM CryptoTools
      {joint-iso-itu-t ds(5) module(1) cryptoTools(42) 9 } WITH SUCCESSORS ;

WRAPPED-PROT ::= TYPE-IDENTIFIER

WrappedProt {WRAPPED-PROT:SupportedProtSet} ::= SEQUENCE {
  id    WRAPPED-PROT.&id({SupportedProtSet}),
  prot  WRAPPED-PROT.&Type({SupportedProtSet}{@id}),
  ... }

WrapperPDU ::= CHOICE {
  handshakeReq       [0]  HandshakeReq,
  handshakeAcc       [1]  HandshakeAcc,
  handshakeWrpRej    [2]  HandshakeWrpRej,
  handshakeProRej    [3]  HandshakeProRej,
  handshakeSecAbort  [4]  HandshakeSecAbort,
  handshakeProAbort  [5]  HandshakeProAbort,
  dtSecAbort         [6]  DtSecAbort,
  applAbort          [7]  ApplAbort,
  releaseReq         [8]  ReleaseReq,
  releaseRsp         [9]  ReleaseRsp,
  dataTransferClient [10] DataTransferClient,
  dataTransferServer [11] DataTransferServer,
  ... }

HandshakeReq ::= Signed{TbsHandshakeReq}

TbsHandshakeReq ::= SEQUENCE {
  version        Version DEFAULT {v1},
  prProt         WRAPPED-PROT.&id ({SupportedProtSet}),
  sigAlg         AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  altSigAlg  [0] AlgorithmIdentifier {{SupportedAltSignatureAlgorithms}} OPTIONAL,
  pkiPath        DER-PkiPath,
  assoID         AssoID,
  time           TimeStamp,
  keyEst         AlgorithmWithInvoke{{SupportedKeyEstablishmentAlgos}},
  altKeyEst  [1] AlgorithmWithInvoke{{SupportedAltKeyEstablishmentAlgos}} OPTIONAL,
```

```
  encr-mode      CHOICE {
    aead       [2] SEQUENCE SIZE (1..MAX) OF
      algo           AlgorithmIdentifier{{SupportedAeadAlgorithms}},
    non-aead   [3] SEQUENCE {
      encr       [0] SEQUENCE SIZE (1..MAX) OF
        algo           AlgorithmIdentifier{{SupportedSymmetricKeyAlgorithms}}
                         OPTIONAL,
      icvAlgID   [1] SEQUENCE SIZE (1..MAX) OF
        algo           AlgorithmIdentifier{{SupportedIcvAlgorithms}} },
    ... },
  attCert        DER-AttributeCertificate OPTIONAL,
  applData   [4] WrappedProt{{SupportedProtSet}} OPTIONAL,
  ... }

Version ::= BIT STRING {
  v1 (0)  -- version 1
  }

DER-PkiPath ::= OCTET STRING
  (CONTAINING PkiPath ENCODED BY der)

DER-AttributeCertificate ::= OCTET STRING
  (CONTAINING AttributeCertificate ENCODED BY der)

der OBJECT IDENTIFIER ::=
  {joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1)}

AssoID ::= INTEGER (0..32767)

TimeStamp ::= GeneralizedTime

SupportedSignatureAlgorithms ALGORITHM ::= {...}

SupportedAltSignatureAlgorithms ALGORITHM ::= {...}

SupportedKeyEstablishmentAlgos ALGORITHM ::= {...}

SupportedAltKeyEstablishmentAlgos ALGORITHM ::= {...}

SupportedAeadAlgorithms ALGORITHM ::= {...}

SupportedSymmetricKeyAlgorithms ALGORITHM ::= {...}

SupportedIcvAlgorithms ALGORITHM ::= {...}

HandshakeAcc ::= Signed{TbsHandshakeAcc}

TbsHandshakeAcc ::= SEQUENCE {
  version        Version DEFAULT {v1},
  sigSel         CHOICE {
    sigAlg         AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
    altSigAlg  [0] AlgorithmIdentifier{{SupportedAltSignatureAlgorithms}} },
  pkiPath        DER-PkiPath,
  assoID         AssoID,
  time           TimeStamp,
  keyEstSel      CHOICE {
    keyEst         AlgorithmWithInvoke{{SupportedKeyEstablishmentAlgos}},
    altKeyEst  [1] AlgorithmWithInvoke{{SupportedAltKeyEstablishmentAlgos}} },
  encr-mode      CHOICE {
    aead       [2] AlgorithmIdentifier{{SupportedAeadAlgorithms}},
    non-aead   [3] SEQUENCE {
      encr       [0] AlgorithmIdentifier{{SupportedSymmetricKeyAlgorithms}} OPTIONAL,
      icvAlgID   [1] AlgorithmIdentifier{{SupportedIcvAlgorithms}} },
    ... },
  attCert        DER-AttributeCertificate OPTIONAL,
  applData   [4] WrappedProt{{SupportedProtSet}} OPTIONAL,
  ... }

HandshakeWrpRej ::= Signed{TbsHandshakeWrpRej}

TbsHandshakeWrpRej ::= SEQUENCE {
```

```
  version        Version DEFAULT {v1},
  sigSel         CHOICE {
    sigAlg         AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
    altSigAlg  [0] AlgorithmIdentifier{{SupportedAltSignatureAlgorithms}} },
  assoID         AssoID,
  time           TimeStamp,
  pkiPath        DER-PkiPath,
  diag           WrpError OPTIONAL,
  ... }

HandshakeProRej ::= Signed{TbsHandshakeProRej}

TbsHandshakeProRej ::= SEQUENCE {
  sigSel         CHOICE {
    sigAlg         AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
    altSigAlg  [0] AlgorithmIdentifier{{SupportedAltSignatureAlgorithms}} },
  assoID         AssoID,
  time           TimeStamp,
  pkiPath        DER-PkiPath,
  applData       WrappedProt{{SupportedProtSet}},
  ... }

HandshakeSecAbort ::= Signed{TbsHandshakeSecAbort}

TbsHandshakeSecAbort ::= SEQUENCE {
  version      Version DEFAULT {v1},
  sigAlg       AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  diag         WrpError OPTIONAL,
  ... }

HandshakeProAbort ::= Signed{TbsHandshakeProAbort}

TbsHandshakeProAbort ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  applData     WrappedProt{{SupportedProtSet}},
  ... }

DtSecAbort ::= Signed{TbsDtSecAbort}

TbsDtSecAbort ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  seq          SequenceNumber,
  diag         WrpError OPTIONAL,
  ... }

ApplAbort ::= Signed{TbsApplAbort}

TbsApplAbort ::= SEQUENCE {
  sigAlg       AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  seq          SequenceNumber,
  applData     WrappedProt{{SupportedProtSet}},
  ... }

ReleaseReq ::= Signed{TbsReleaseReq}

TbsReleaseReq ::= SEQUENCE {
  version      Version DEFAULT {v1},
  sigAlg       AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
  assoID       AssoID,
```

```
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  ... }

ReleaseRsp ::= Signed{TbsReleaseRsp}

TbsReleaseRsp ::= SEQUENCE {
  version      Version DEFAULT {v1},
  sigAlg       AlgorithmIdentifier{{SupportedSignatureAlgorithms}},
  assoID       AssoID,
  time         TimeStamp,
  pkiPath      DER-PkiPath,
  ... }

DataTransferClient ::= CHOICE {
  aead     [0] DataTransferClientAE,
  non-aead [1] DataTransferClientNEA,
  ... }

DataTransferClientAE ::= AUTHEN-ENCRYPT{AadClientAE, WRAPPED-PROT.&Type}

AadClientAE ::= SEQUENCE {
  COMPONENTS OF    AadClient,
  encInvoke    [3] AlgoInvoke{{SupportedAeadAlgorithms}} OPTIONAL,
  ... }

DataTransferClientNEA ::= ICV-Invoke{TbpDataTransferClient}

TbpDataTransferClient ::= SEQUENCE {
  COMPONENTS OF    AadClient,
  encEnvoke    [3] AlgoInvoke{{SupportedSymmetricKeyAlgorithms}} OPTIONAL,
  conf             CHOICE {
    clear        [4] WrappedProt{{SupportedProtSet}},
    protected    [5] ENCIPHERED{WRAPPED-PROT.&Type},
    ... },
  ... }

AadClient ::= SEQUENCE {
  invokeID [0] InvokeID OPTIONAL,
  assoID       AssoID,
  time         TimeStamp,
  seq          SequenceNumber,
  keyEst   [2] AlgoInvoke{{SupportedKeyEstablishmentAlgos}} OPTIONAL }

InvokeID ::= OCTET STRING (SIZE (6))

SequenceNumber ::= INTEGER (0..2147483647)

DataTransferServer ::= CHOICE {
  aead     [0] DataTransferServerAE,
  non-aead [1] DataTransferServerNEA,
  ... }

DataTransferServerAE ::= AUTHEN-ENCRYPT{AadServerAE, WRAPPED-PROT.&Type}

AadServerAE ::= SEQUENCE {
  COMPONENTS OF AadServer,
  encInvoke [3] AlgoInvoke{{SupportedAeadAlgorithms}} OPTIONAL,
  ... }

DataTransferServerNEA ::= ICV-Invoke{TbpDataTransferServer}

TbpDataTransferServer ::= SEQUENCE {
  COMPONENTS OF     AadServer,
  encInvoke     [3] AlgoInvoke{{SupportedSymmetricKeyAlgorithms}} OPTIONAL,
  conf              CHOICE {
    clear         [4] WrappedProt{{SupportedProtSet}},
    protected     [5] ENCIPHERED{WRAPPED-PROT.&Type},
    ... },
  ... }
```

```
AadServer ::= SEQUENCE {
  invokeID   [0] InvokeID OPTIONAL,
  assoID         AssoID,
  time           TimeStamp,
  seq            SequenceNumber,
  reqRekey   [1] BOOLEAN DEFAULT FALSE,
  changedKey [2] BOOLEAN DEFAULT FALSE }

WrpError ::= ENUMERATED {
  protocol-error                          (0),
  invalid-signatureAlgorithm              (1),
  unexpected-version                      (2),
  protected-protocol-not-supported        (3),
  duplicate-assoID                        (4),
  invalid-time-value                      (5),
  key-estab-algorithm-not-supported       (6),
  encr-mode-aead-not-supported            (7),
  encryption-not-supported                (8),
  encryption-required                     (9),
  aead-algorithms-not-supported           (10),
  aead-is-required                        (11),
  symmetricKey-algorithms-not-supported   (12),
  icv-algorithms-not-supported            (13),
  invalid-attribute-certificate           (14),
  alt-signature-not-allowed               (15),
  only-one-version                        (16),
  invalid-key-estab-algorithm             (17),
  invalid-alt-key-estab-algorithm         (18),
  invalid-aead-algorithm                  (19),
  aead-not-allowed                        (20),
  invalid-symmetricKey-algorithm          (21),
  invalid-icv-algorithm                   (22),
  dynamic-aead-algo-parms-required        (23),
  invalid-dynamic-aead-algo-parms         (24),
  dynamic-aead-algo-parms-not-required    (25),
  dynamic-symKey-algo-parms-required      (26),
  invalid-dynamic-symKey-algo-parms       (27),
  dynamic-symKey-algo-parms-not-required (28),
  dynamic-icv-algo-parms-required         (29),
  invalid-dynamic-icv-algo-parms          (30),
  dynamic-icv-algo-parms-not-required     (31),
  unexpected-invokeID-received            (32),
  rekey-out-of-sequence                   (33),
  invalid-dynamic-keyEst-algo-parms       (34),
  changedKey-out-of-sequence              (35),
  ... }

END -- Wrapper
*
```

Annex C

Protected protocol interface to the wrapper protocol

(This annex forms an integral part of this Recommendation | International Standard.)

As specified in clause 8.8, a protected protocol is defined as an information object of the **WRAPPER-PROT** information object class. This means that a particular protected protocol may be specified as follows.

```
<identifier> WRAPPED-PROT ::= {
              <The top-level APDU of the protected protocol>
  IDENFIED BY <object identifier for protected protocol>
```

A protected protocol is then represented by a data value being an instance of the **WrapperProt** data type specified in clause 8.8. This implies that the protected protocol is a sequence of the identifying object identifier and the chosen PrPDU from the top-level of the APDU.

For a specific purpose, e.g., creating an implementation, an ASN.1 module, named **ProtProtocols** shall be included in the set of ASN.1 modules to be compiled. The structure of this module is shown in the following.

This annex defines three different protocols potentially to be protected. For each of them, the top-level APDU is imported from the relevant module.

The object set **SupportedProtSet** is then imported by the wrapper protocol allowing either of the protocols in the object set to be protected. To ease implementation only a single information object should be included.

```
ProtProtocols {joint-iso-itu-t ds(5) module(1) protProtocols(47) 9}
IMPLICIT TAGS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All

IMPORTS

  id-wrprot
    FROM
      PKI-Stub {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9} WITH SUCCESSORS

  WRAPPED-PROT
    FROM Wrapper
      {joint-iso-itu-t ds(5) module(1) wrapper(43) 9} WITH SUCCESSORS

  AvlProt
    FROM AVL-management
      {joint-iso-itu-t ds(5) module(1) avl-management(44) 9} WITH SUCCESSORS

  CasubProt
    FROM CaSubscription
      {joint-iso-itu-t ds(5) module(1) caSubscription(45) 9} WITH SUCCESSORS

  TBprot
    FROM TrustBroker
      {joint-iso-itu-t ds(5) module(1) trustBroker(46) 9} WITH SUCCESSORS;


avlProt WRAPPED-PROT ::= {
                AvlProt
  IDENTIFIED BY  id-avlprot }

casubProt WRAPPED-PROT ::= {
                CasubProt
  IDENTIFIED BY  id-casubprot }

tbprot WRAPPED-PROT ::= {
                TBprot
  IDENTIFIED BY  id-tbprot }

SupportedProtSet WRAPPED-PROT ::= {avlProt | casubProt | tbprot }
```

```
id-avlprot          OBJECT IDENTIFIER ::= {id-wrprot 0}
id-casubprot        OBJECT IDENTIFIER ::= {id-wrprot 1}
id-tbprot           OBJECT IDENTIFIER ::= {id-wrprot 2}

END -- ProtProtocols
```

Annex D

Cryptographic algorithms

(This annex forms an integral part of this Recommendation | International Standard.)

This annex includes all the ASN.1 type, value and information object definitions for the use of cryptographic algorithms in the form of the ASN.1 module **GenAlgo**.

```
GenAlgo {joint-iso-itu-t ds(5) module(1) genAlgo(48) 9}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All

IMPORTS

  ALGORITHM, id-algo
    FROM
      PKI-Stub {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9} WITH SUCCESSORS ;

id-algo-mca  OBJECT IDENTIFIER ::= {id-algo 1} -- multiple-cryptographic algorithms
id-algo-ska  OBJECT IDENTIFIER ::= {id-algo 2} -- symmetric-key algorithms
id-algo-aead OBJECT IDENTIFIER ::= {id-algo 3} -- authenticated encryption with asso data
id-algo-pka  OBJECT IDENTIFIER ::= {id-algo 4} -- public-key algorithms
id-algo-ha   OBJECT IDENTIFIER ::= {id-algo 5} -- hash algorithms
id-algo-dsa  OBJECT IDENTIFIER ::= {id-algo 6} -- digital dignature algorithms
id-algo-kea  OBJECT IDENTIFIER ::= {id-algo 7} -- key establishment algorithms


-- Key establishment algorithms

dhModpGr14Hkdf256Algo ALGORITHM ::= {
  PARMS         Group14
  DYN-PARMS     Payload14
  IDENTIFIED BY id-algo-dhModpGr14Hkdf256Algo }

Group14 ::= INTEGER (14)

Payload14 ::= SEQUENCE {
  dhPublicKey OCTET STRING (SIZE (256)),
  nonce       OCTET STRING (SIZE (32)),
  ... }

dhModpGr23Hkdf256Algo ALGORITHM ::= {
  PARMS         Group23
  DYN-PARMS     Payload23
  IDENTIFIED BY id-algo-dhModpGr23Hkdf256Algo }

Group23 ::= INTEGER (23)

Payload23 ::= SEQUENCE {
  dhPublicKey OCTET STRING (SIZE (512)),
  nonce       OCTET STRING (SIZE (32)),
  ... }

dhModpGr28Hkdf256Algo ALGORITHM ::= {
  PARMS         Group28
  DYN-PARMS     Payload28
  IDENTIFIED BY id-algo-dhModpGr28Hkdf256Algo }

Group28 ::= INTEGER (28)

Payload28 ::= SEQUENCE {
  dhPublicKey OCTET STRING (SIZE (512)),
  nonce       OCTET STRING (SIZE (32)),
  ... }
```

```
-- Object identifier allocation

id-algo-dhModpGr14Hkdf256Algo       OBJECT IDENTIFIER ::= {id-algo-kea 1}
id-algo-dhModpGr15Hkdf384Algo       OBJECT IDENTIFIER ::= {id-algo-kea 2}
id-algo-dhModpGr16Hkdf512Algo       OBJECT IDENTIFIER ::= {id-algo-kea 3}
id-algo-dhModpGr17Hkdf768Algo       OBJECT IDENTIFIER ::= {id-algo-kea 4}
id-algo-dhModpGr18Hkdf1024Algo      OBJECT IDENTIFIER ::= {id-algo-kea 5}

id-algo-dhModpGr23Hkdf256Algo       OBJECT IDENTIFIER ::= {id-algo-kea 10}

id-algo-dhModpGr28Hkdf256Algo       OBJECT IDENTIFIER ::= {id-algo-kea 15}

END -- GenAlgo
```

Annex E

Authorization and validation list management in ASN.1

(This annex forms an integral part of this Recommendation | International Standard.)

This annex includes all the ASN.1 type, value and information object definitions for the AVMP in the form of the ASN.1 module **AVL-management**.

NOTE – This module was part of the **PkiPMIProtocolSpecifications** in Rec. ITU-T X.509 (2016) | ISO/IEC 9594-8:2017.

```
AVL-management {joint-iso-itu-t ds(5) module(1) avl-management(44) 9}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All

IMPORTS
/*
  -- from Rec. ITU-T X.501 | ISO/IEC 9594-2

  Attribute{}, SupportedAttributes
    FROM InformationFramework
      {joint-iso-itu-t ds(5) module(1) informationFramework(1) 9} WITH SUCCESSORS

  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8

  Certificate, CertAVL, AvlSerialNumber
    FROM AuthenticationFramework
      {joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 9}
*/

  -- From x510-import

  Attribute{}, AvlSerialNumber, CertAVL, Certificate, SupportedAttributes
    FROM PKI-Stub
      {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9} WITH SUCCESSORS

  Version
    FROM Wrapper
      {joint-iso-itu-t ds(5) module(1) wrapper(43) 9} WITH SUCCESSORS ;

-- PDU types

AvlProt ::= CHOICE {
  initReq       [0]  InitializationRec,
  initAcc       [1]  InitializationAcc,
  initRej       [2]  InitializationRej,
  initAbt       [3]  InitializationAbort,
  certReq       [4]  CertReq,
  certRsp       [5]  CertRsp,
  addAvlReq     [6]  AddAvlReq,
  addAvlRsp     [7]  AddAvlRsp,
  replaceAvlReq [8]  ReplaceAvlReq,
  replaceAvlRsp [9]  ReplaceAvlRsp,
  deleteAvlReq  [10] DeleteAvlReq,
  deleteAvlRsp  [11] DeleteAvlRsp,
  abortAVL      [12] AbortAVL,
  ... }

InitializationRec ::= SEQUENCE {
  version    Version,
  ... }

InitializationAcc ::= SEQUENCE {
  version    Version,
  ... }
```

```
InitializationRej ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion     (0),
    ... },
  ... }

InitializationAbort ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion       (0),
    onlySingleVersionAllowed (1),
    ... },
  ... }

CertReq ::= SEQUENCE {
  invokeID  InvokeID,
  ... }

InvokeID ::= INTEGER (0..127)

CertRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success [0]  CertOK,
    failure [1]  CertErr,
    ... },
  ... }

CertOK ::= SEQUENCE {
  dhCert  Certificate,
  ... }

CertErr ::= SEQUENCE {
  notOK  AVMP-error,
  note   Notifications OPTIONAL,
  ... }

Notifications ::= SEQUENCE SIZE (1..MAX) OF Attribute {{SupportedAttributes}}

AddAvlReq ::= SEQUENCE {
  invokeID     InvokeID,
  certlist     CertAVL,
  ... }

AddAvlRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success [0]  AddAvlOK,
    failure [1]  AddAvlErr,
    ... },
  ... }

AddAvlOK ::= SEQUENCE {
  ok     NULL,
  ... }

AddAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }

ReplaceAvlReq ::= SEQUENCE {
  invokeID     InvokeID,
  old          AvlSerialNumber OPTIONAL,
  new          CertAVL,
  ... }

ReplaceAvlRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success [0]  RepAvlOK,
    failure [1]  RepAvlErr,
    ... },
```

```
  ... }

RepAvlOK ::= SEQUENCE {
  ok     NULL,
  ... }

RepAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }

DeleteAvlReq ::= SEQUENCE {
  invokeID     InvokeID,
  avl-Id       AvlSerialNumber OPTIONAL,
  ... }

DeleteAvlRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success [0]  DelAvlOK,
    failure [1]  DelAvlErr,
    ... },
  ... }

DelAvlOK ::= SEQUENCE {
  ok     NULL,
  ... }

DelAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }

AbortAVL ::= SEQUENCE {
  invokeID     InvokeID,
  reason       AVMP-error,
  ... }

AVMP-error ::= ENUMERATED {
  noReason                           (0),
  protocolError                      (1),
  duplicateAVL                       (2),
  missingAvlComponent                (3),
  invalidAvlVersion                  (4),
  notAllowedForConstrainedAVLEntity  (5),
  constrainedRequired                (6),
  nonConstrainedRequired             (7),
  unsupportedCriticalEntryExtension (8),
  unsupportedCriticalExtension      (9),
  maxAVLsExceeded                    (10),
  unknownAVL                         (11),
  ... }

END -- AVL-management
```

Annex F

Certification authority subscription in ASN.1

(This annex forms an integral part of this Recommendation | International Standard.)

This annex includes all the ASN.1 type, value and information object definitions used by the CASP in the form of the ASN.1 module **CaSubsription**.

NOTE – This module was part of the **PkiPMIProtocolSpecifications** in Rec. ITU-T X.509 (2016) | ISO/IEC 9594-8:2017.

```
CaSubscription {joint-iso-itu-t ds(5) module(1) caSubscription(45) 9}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All

IMPORTS
/*
  -- from Rec. ITU-T X.501 | ISO/IEC 9594-2

  Name
    FROM InformationFramework
      {joint-iso-itu-t ds(5) module(1) informationFramework(1) 9} WITH SUCCESSORS

  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8

  Certificate, CertificateSerialNumber
    FROM AuthenticationFramework
      {joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 9}

  CRLReason
    FROM CertificateExtensions
      {joint-iso-itu-t ds(5) module(1) certificateExtensions(26) 9}
*/

  Certificate, CertificateSerialNumber, CRLReason, Name
    FROM PKI-Stub
      {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9}

  -- from Rec. ITU-T X.510 | ISO/IEC 9594-11

  Version
    FROM Wrapper
      {joint-iso-itu-t ds(5) module(1) wrapper(43) 9} WITH SUCCESSORS ;

CasubProt ::= CHOICE {
  initReq             [0]  InitializationRec,
  initAcc             [1]  InitializationAcc,
  initRej             [2]  InitializationRej,
  initAbt             [3]  InitializationAbort,
  certSubscribeReq    [4]  CertSubscribeReq,
  certSubscribeRsp    [5]  CertSubscribeRsp,
  certUnsubscribeReq  [6]  CertUnsubscribeReq,
  certUnsubscribeRsp  [7]  CertUnsubscribeRsp,
  certReplaceReq      [8]  CertReplaceReq,
  certReplaceRsp      [9]  CertReplaceRsp,
  certUpdateReq       [10] CertUpdateReq,
  certUpdateRsp       [11] CertUpdateRsp,
  cAsubscribeAbort    [12] CAsubscribeAbort,
  ... }

InitializationRec ::= SEQUENCE {
  version    Version,
  ... }

InitializationAcc ::= SEQUENCE {
  version    Version,
```

```
  ... }

InitializationRej ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion     (0),
    ... },
  ... }

InitializationAbort ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion       (0),
    onlySingleVersionAllowed (1),
    ... },
  ... }

CertSubscribeReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs        SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }

InvokeID ::= INTEGER (0..127)

CertSubscribeRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
    success       [0]  CertSubscribeOK,
    failure       [1]  CertSubscribeErr,
    ... },
  ... }

CertSubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok       [0] SEQUENCE {
    cert         Certificate,
    status       CertStatus,
    revokeReason CRLReason OPTIONAL,
    ... },
  not-ok   [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CertStatus ::= ENUMERATED {
  good    (0),
  revoked (1),
  on-hold (2),
  expired (3),
  ... }

CASP-CertStatusCode ::= ENUMERATED {
  noReason       (1),
  unknownCert    (2),
  ... }

CertSubscribeErr ::= SEQUENCE {
  code       CASP-error,
  ... }

CertUnsubscribeReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs  SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }

CertUnsubscribeRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result       CHOICE {
```

```
    success       [0]  CertUnsubscribeOK,
    failure       [1]  CertUnsubscribeErr,
    ... },
  ... }

CertUnsubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok       [0] SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  not-ok   [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CertUnsubscribeErr ::= SEQUENCE {
  code         CASP-error,
  ... }

CertReplaceReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs         SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    old           CertificateSerialNumber,
    new           Certificate,
    ... },
  ... }

CertReplaceRsp ::= SEQUENCE {
  invokeID     InvokeID,
  result        CHOICE {
    success       [0]  CertReplaceOK,
    failure       [1]  CertReplaceErr,
    ... },
  ... }

CertReplaceOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok        [0] SEQUENCE {
    issuer        Name,
    serialNumber  CertificateSerialNumber,
    ... },
  not-ok    [1] SEQUENCE {
    status        CASP-CertStatusCode,
    ... },
  ... }

CertReplaceErr ::= SEQUENCE {
  code          CASP-error,
  ... }

CertUpdateReq ::= SEQUENCE {
  invokeID     InvokeID,
  certs  SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    certStatus   CertStatus,
    ... },
  ... }

CertUpdateRsp ::= SEQUENCE {
  invokeID      InvokeID,
  result        CHOICE {
    success       [0]  CertUpdateOK,
    failure       [1]  CertUpdateErr,
    ... },
  ... }

CertUpdateOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok        [0] SEQUENCE {
    subject       Name,
    serialNumber  CertificateSerialNumber,
    ... },
```

```
  not-ok    [1] SEQUENCE {
    status        CASP-CertStatusCode,
    ... },
  ... }

CertUpdateErr ::= SEQUENCE {
  code          CASP-error,
  ... }

CAsubscribeAbort ::= SEQUENCE {
  invokeID     InvokeID,
  reason       CASP-error,
  ... }

CASP-error ::= ENUMERATED {
  noReason                      (0),
  unknownContentType            (1),
  unsupportedWLMPversion        (2),
  missingContent                (3),
  missingContentComponent       (4),
  invalidContentComponent       (5),
  sequenceError                 (6),
  unknownSubject                (7),
  unknownCert                   (8),
  ... }

END -- CaSubsription
```

Annex G

Trust broker in ASN.1

(This annex forms an integral part of this Recommendation | International Standard.)

This annex includes all the ASN.1 type, value and information object definitions used by the trust broker protocol in the form of the ASN.1 module **TrustBroker**.

NOTE – This module was part of the **PkiPMIProtocolSpecifications** in Rec. ITU-T X.509 (2016) | ISO/IEC 9594-8:2017.

```
TrustBroker {joint-iso-itu-t ds(5) module(1) trustBroker(46) 9}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All

IMPORTS

  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8
/*
  PKCertIdentifier
    FROM AuthenticationFramework
      {joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 9} WITH SUCCESSORS
*/
  PKCertIdentifier
    FROM PKI-Stub
      {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9} WITH SUCCESSORS

  Version
    FROM Wrapper
      {joint-iso-itu-t ds(5) module(1) wrapper(43) 9} WITH SUCCESSORS ;

 -- PDU types

TBprot ::= CHOICE {
  initReq     [0] InitializationReq,
  initAcc     [1] InitializationAcc,
  initRej     [2] InitializationRej,
  initAbt     [3] InitializationAbort,
  tBrequest   [4] TBrequest,
  tBresponse  [5] TBresponse,
  ... }

InitializationReq ::= SEQUENCE {
  version    Version,
  ... }

InitializationAcc ::= SEQUENCE {
  version    Version,
  ... }

InitializationRej ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersions     (0),
    ... },
  ... }

InitializationAbort ::= SEQUENCE {
  diag        ENUMERATED {
    unsupportedVersion       (0),
    onlySingleVersionAllowed (1),
    ... },
  ... }

TBrequest ::= CHOICE {
  caCert      [0] PKCertIdentifier,
  subjectCert [1] PKCertIdentifier,
```

```
  ... }

TBresponse ::= CHOICE {
  success [0]  TBOK,
  failure [1]  TBerror,
  ... }

TBOK ::= SEQUENCE {
  levelOfAssurance  [0]  INTEGER (0..100),
  confidenceLevel   [1]  INTEGER (0..100),
  validationTime    [2]  UTCTime,
  info              [3]  UTF8String  OPTIONAL,
  ... }

TBerror ::= SEQUENCE {
  code        ENUMERATED {
    caCertInvalid        (1),
    unknownCert          (2),
    unknownCertStatus    (3),
    subjectCertRevoked   (4),
    incorrectCert        (5),
    contractExpired      (6),
    pathValidationFailed (7),
    timeOut              (8),
    other                (99),
    ... },
  diagnostic  UTF8String OPTIONAL,
  ... }

END -- Trustbroker
```

Annex H

Migration of cryptographic algorithms

(This annex does not form an integral part of this Recommendation | International Standard.)

H.1 Introduction

This annex applies to protocols in general and not only to the protocol defined by this Specification.

When moving from one cryptographic algorithm to an assumingly stronger cryptographic algorithm of the same type, not everybody will migrate at the same time, which potentially will give interworking problems. During a migration period, it should be possible for some entities to use a new stronger cryptographic algorithm while others for a while may keep using the old one, while at the same time maintaining interworking among all entities.

The migration of cryptographic algorithms occurs in different situations.

a) Use of cryptographic algorithms is negotiated before their use. This is the case where there is a connection-oriented application layer protocol, i.e. a protocol specification where there is a handshake procedure followed by a data transfer phase. Cryptographic algorithms used during the data transfer phase can then be negotiated during the handshake procedure. This can further be divided into:
   - measures taken for new application protocols;
   - measures taken for existing application protocols.

b) Cryptographic algorithms that are used in the same APDU in which they are specified. This can again be divided into two cases:
   - measures taken for new application protocols;
   - measures taken for existing application protocols.

   This may further be divided into migration of digital signature algorithms with associated digital signatures and migration of other types of cryptographic algorithms.

H.2 Negotiation of cryptographic algorithms

H.2.1 Cryptographic negotiation for new protocols

The client may use the handshake exchange to propose cryptographic algorithms to be used during the subsequent data transfer phase allowing for negotiation as described in the following.

The following component may be included in a handshake request.

```
<identifier>   SEQUENCE SIZE (1..MAX) OF
   AlgorithmIdentifier {{<algorithm object set of a specific information object class>}},
```

This component will allow the client to specify a sequence of supported algorithms of the same type of information objects, e.g., a sequence of hash algorithms. For migration purposes, at last two algorithms shall be included. The first one in the sequence shall then be the alternative algorithm, i.e., the algorithm to which migration is wanted, while the second algorithm should be the native algorithm, i.e., the algorithm from which migration is wanted. This assumes that the client supports and is ready to migrate to the alternative algorithm. Otherwise, the client will only supply the native algorithm.

The server shall then in the handshake accept include a single algorithm as follows.

```
<identifier>   AlgorithmIdentifier
  {{<algorithm object set of specific a specific object class>}},
```

Assuming that the client has supplied an alternative algorithm together with the native algorithm in the sequence-of, the server shall select the first cryptographic algorithm it supports of those proposed by client. If this is the first algorithm in the sequence-of, i.e., the alternative algorithm, then both the client and the server have migrated. They may now for future mutual communication use what was the alternative algorithm as the new native algorithm.

H.2.2 Cryptographic negotiation for existing protocols

If an existing protocol already in the handshake procedure has cryptographic algorithm definitions as described in clause H.2.1, then that subclause applies, although there may be some updates to the procedure.

If the original handshake request only specifies a single cryptographic algorithm to be used during the data transfer phase, there are two approaches depending on whether ASN.1 extension marks are used by the protocol specification.

In the example shown in Figure H.1, the client suggests a single symmetric-key algorithm to be used during the data transfer phase. As ASN.1 extension marks are supported, an alternative algorithm specification may then be placed after the extension mark.

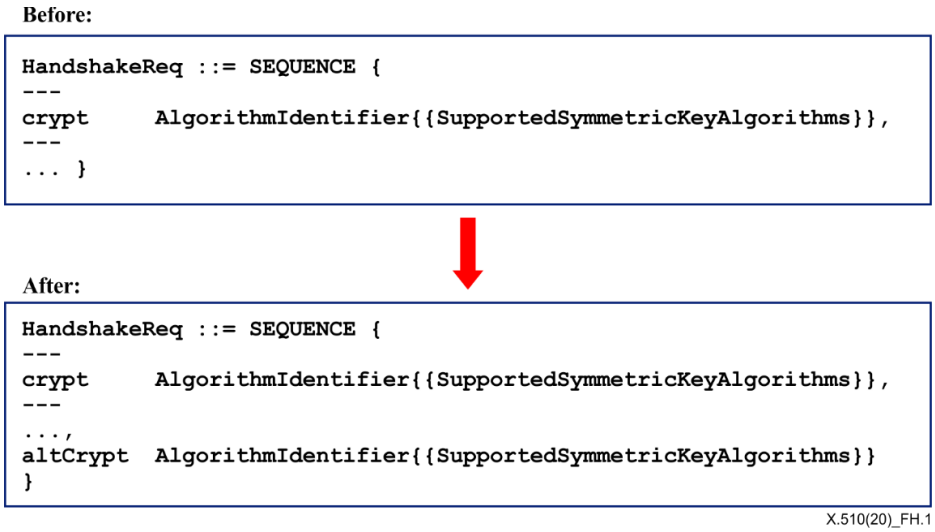

Figure H.1 – Alternative algorithm with extension marks supported

This approach has the advantage that an implementation that supports extension marks, but does not understand the component with the alternative encryption algorithm, will continue to function. The approach has the disadvantage that it requires an update to the protocol specification in question.

Another technique is to use the appropriate multiple-cryptographic algorithm specified in clause 6.3 allowing for specification of several cryptographic algorithms within a single outer algorithm specification.

This technique has the advantage that the protocol specification does not need to be changed. The disadvantage is that implementations will fail if they do not support the multiple-cryptographic algorithm in question.

As described in the foregoing, the client specifies the algorithm according to preference by placing the alternative algorithm as the first algorithm and the native algorithm as the second algorithm. The server then selects the first one it supports.

H.3 Non-negotiable digital signature algorithms

H.3.1 General

Digital signature algorithms and digital signatures are generally used as part of handshake procedure and cannot therefore be negotiated before their use.

Figure H.2 depicts a structure generally used. Some information requires a digital signature. The digital signature algorithm to be used is place inside the data that has to be signed and the digital signature is then created over the data to be signed and appended to the data.

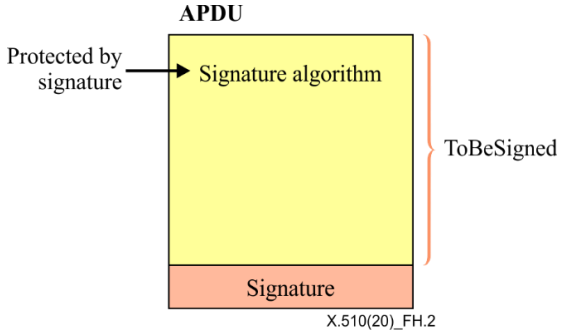

Figure H.2 – Use of digital signatures

H.3.2 Duplicate signatures for new protocols

Figure H.3 depicts a way of providing alternative digital signature algorithm together with an alternative digital signature for new protocols. The alternative digital signature algorithm may my placed adjacent to the native digital signature algorithm within the signed area.

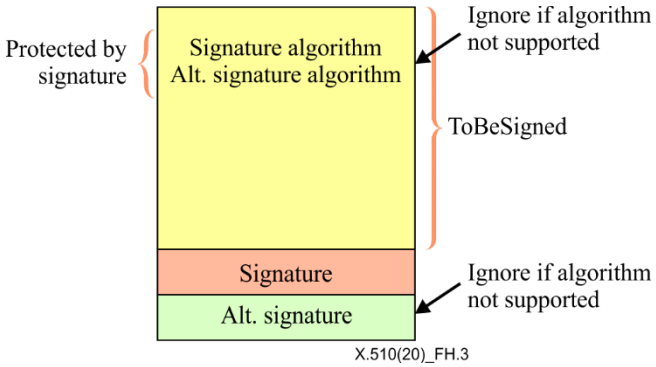

Figure H.3 – Duplicate signatures for new protocols

This is further described in Figure H.4, where the handshake request holds two digital signature algorithms, while the response only holds one. If the server supports the alternative digital signature algorithm, the server will select it for inclusion in the response. Otherwise, it will select the native digital signature algorithm.

The response only holds a single digital signature generated according to the selected digital signature algorithm.

If the server selects the alternative signature algorithm, both the client and the server have migrated to the alternative signature and in future may use the alternative digital signature algorithm as the new native digital signature algorithm.

Request:

```
ReqApdu ::= SEQUENCE {
---
 sigAlg        AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
 altSigAlg [0] AlgorithmIdentifier {{SupportedSignatureAlgorithms}} OPTIONAL,
 ---
 signature     BIT STRING,
 altSignature  BIT STRING OPTIONAL }
```

Response:

```
RspApdu ::= SEQUENCE {
---
 sigSel          CHOICE {
   sigAlg          AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
   altSigAlg  [0] AlgorithmIdentifier {{SupportedSignatureAlgorithms}} },
 ---
signature        BIT STRING }
```

X.510(20)_FH.4

Figure H.4 – Negotiation of digital signature algorithm

The **`Signed`** parametrized data specified clause 6.5.2 may be used for the digital signature process.

H.3.3 Duplicate signatures for existing protocols

Generally, it is not possible to specify how existing protocols may be updated to provide a migration path. This will depend on how the protocol is designed. The different tools for establishing a migration path defined in clauses 6.3 and 6.5 may be used.

In some cases, the **`SIGNED`** parameterized data type defined in clause 6.2.1 of Rec. ITU-T X.509 | ISO/IEC 9594-8 may be utilized when it has originally been used in its simpler form, i.e., where the **`altAlgorithmIdentifier`** and **`altSignature`** components are absent. This is illustrated in Figure H.5.

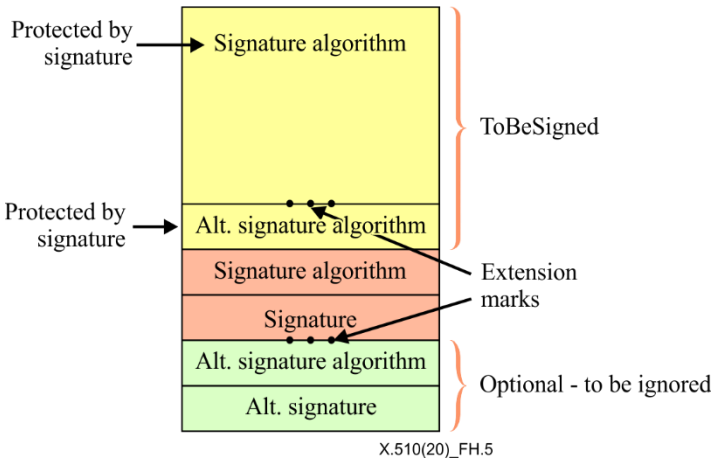

Figure H.5 – Duplicate signatures for existing protocols

Providing an alternative digital signature for an existing protocol may be divided into whether the protocol supports ASN.1 extension marks are not.

Figure H.5 represents the situation where a protocol in general has included extension marks. This is a similar situation to that described in clause H.3.2, with the exception that the alternative digital signature algorithm is placed after the extension mark and therefore will be ignored if the server does not expect that component. In this case, the server shall use the alternative digital signature algorithm if it is supported for the native digital signature algorithm component. Likewise, the server shall only include a single signature.

If ASN.1 extension marks are not supported, a technique like that depicted in Figure H.2 may be used. The **`multipleSignaturesAlgo`** algorithm specification given in clause 6.2 may be used within the protected area, while multiple signatures may be specified by use of the **`MULTY-SIGNED{ToBeSigned}`** data type defined in clause 6.3.

This technique does not require any changes to the existing protocol, although some updated procedures will have to be observed. It will require changes to implementations.

Annex I

Auxiliary specifications

(This annex does not form an integral part of this Recommendation | International Standard.)

This annex reproduces the ASN.1 specification from other Specifications in the ITU X.500 Series of Recommendations | ISO/IEC 9594-all parts.

```
PKI-Stub {joint-iso-itu-t ds(5) module(1) pki-stub(999) 9}
DEFINITIONS ::=
BEGIN

id-wrprot            OBJECT IDENTIFIER ::= wrapperProtocolType
wrapperProtocolType  OBJECT IDENTIFIER ::= {ds 43}
ds                   OBJECT IDENTIFIER ::= {joint-iso-itu-t ds(5)}
id-algo              OBJECT IDENTIFIER ::= algorithms
algorithms           OBJECT IDENTIFIER ::= {ds 44}

ALGORITHM ::= CLASS {
  &Type          OPTIONAL,
  &DynParms      OPTIONAL,
  &id            OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  [PARMS         &Type]
  [DYN-PARMS     &DynParms ]
  IDENTIFIED BY  &id }

AlgorithmWithInvoke{ALGORITHM:SupportedAlgorithms} ::= SEQUENCE {
  algorithm       ALGORITHM.&id({SupportedAlgorithms}),
  parameters  [0] ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL,
  dynamParms  [1] ALGORITHM.&DynParms({SupportedAlgorithms}{@algorithm}) OPTIONAL,
  ... }

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

AlgoInvoke{ALGORITHM:SupportedAlgorithms} ::=
    ALGORITHM.&DynParms({SupportedAlgorithms})

HASH{ToBeHashed} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  hashValue            BIT STRING,
  ... }

SupportedAlgorithms ALGORITHM ::= {...}

SIGNED{ToBeSigned} ::= SEQUENCE {
  toBeSigned              ToBeSigned,
  algorithmIdentifier     AlgorithmIdentifier{{SupportedAlgorithms}},
  signature               BIT STRING,
  ...,
  altAlgorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}} OPTIONAL,
  altSignature            BIT STRING OPTIONAL
  } (WITH COMPONENTS {..., altAlgorithmIdentifier PRESENT, altSignature PRESENT } |
     WITH COMPONENTS {..., altAlgorithmIdentifier ABSENT,  altSignature ABSENT } )

FingerPrint {ToBeFingerprinted} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  fingerprint          BIT STRING,
  ... }

PkiPath ::= SEQUENCE SIZE (1..MAX) OF Certificate

Certificate ::= SIGNED{TBSCertificate}
```

```
TBSCertificate ::= SEQUENCE {
  version                  [0]  Version DEFAULT v1,
  serialNumber                  CertificateSerialNumber,
  signature                     AlgorithmIdentifier{{SupportedAlgorithms}},
  issuer                        Name,
  validity                      Validity,
  subject                       Name,
  subjectPublicKeyInfo          SubjectPublicKeyInfo,
  issuerUniqueIdentifier   [1] IMPLICIT UniqueIdentifier OPTIONAL,
  ...,
  --[[2:  if present, version shall be v2 or v3
  subjectUniqueIdentifier  [2] IMPLICIT UniqueIdentifier OPTIONAL--]]--,
  --[[3:  if present, version shall be v2 or v3
  extensions               [3]  Extensions OPTIONAL --]]
  -- If present, version shall be v3]]
 } (CONSTRAINED BY { -- shall be DER encoded -- } )

Version ::= INTEGER {v1(0), v2(1), v3(2)}

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {
  notBefore  Time,
  notAfter   Time,
  ... }

SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm         AlgorithmIdentifier{{SupportedAlgorithms}},
  subjectPublicKey  PublicKey,
  ... }

PublicKey ::= BIT STRING

Time ::= CHOICE {
  utcTime          UTCTime,
  generalizedTime  GeneralizedTime }

UniqueIdentifier ::= BIT STRING

Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension

-- For those extensions where ordering of individual extensions within the SEQUENCE is
-- significant, the specification of those individual extensions shall include the
-- rules for the significance of the order therein

Extension ::= SEQUENCE {
  extnId     EXTENSION.&id({ExtensionSet}),
  critical   BOOLEAN DEFAULT FALSE,
  extnValue  OCTET STRING
    (CONTAINING EXTENSION.&ExtnType({ExtensionSet}{@extnId})
       ENCODED BY der),
  ... }

der OBJECT IDENTIFIER ::=
  {joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1)}

ExtensionSet EXTENSION ::= {...}

EXTENSION ::= CLASS {
  &id           OBJECT IDENTIFIER UNIQUE,
  &ExtnType }
WITH SYNTAX {
  SYNTAX        &ExtnType
  IDENTIFIED BY &id }

Name ::= CHOICE { -- only one possibility for now -- rdnSequence  RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET SIZE (1..MAX) OF AttributeTypeAndValue
```

```
DistinguishedName ::= RDNSequence

AttributeTypeAndValue ::= SEQUENCE {
  type                  ATTRIBUTE.&id, --({SupportedAttributes}),
  value                 ATTRIBUTE.&type, --({SupportedAttributes}{@type}),
  ... }

SupportedAttributes ATTRIBUTE ::= {...}

ATTRIBUTE ::= CLASS {
  &type                     UTF8String,
  &id                       OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  WITH SYNTAX               &type
  ID                        &id }

Attribute {ATTRIBUTE:SupportedAttributes} ::= SEQUENCE {
  type                ATTRIBUTE.&id({SupportedAttributes}),
  values              SET SIZE (0..MAX) OF ATTRIBUTE.&type({SupportedAttributes}{@type}),
  ... }

AttributeCertificate ::= SIGNED{TBSAttributeCertificate}

TBSAttributeCertificate ::= SEQUENCE {
  version                 AttCertVersion, -- version is v2
  holder                  Holder,
  issuer                  AttCertIssuer,
  signature               AlgorithmIdentifier{{SupportedAlgorithms}},
  serialNumber            CertificateSerialNumber,
  attrCertValidityPeriod  AttCertValidityPeriod,
  attributes              SEQUENCE OF Attribute{{SupportedAttributes}},
  issuerUniqueID          UniqueIdentifier OPTIONAL,
  ...,
  ...,
  extensions              Extensions OPTIONAL
 }  (CONSTRAINED BY { -- shall be DER encoded -- } )

AttCertVersion ::= INTEGER {v2(1)}

Holder ::= SEQUENCE {
  baseCertificateID  [0]  IssuerSerial OPTIONAL,
  entityName         [1]  GeneralNames OPTIONAL,
  objectDigestInfo   [2]  ObjectDigestInfo OPTIONAL }
  (WITH COMPONENTS {..., baseCertificateID  PRESENT } |
   WITH COMPONENTS {..., entityName  PRESENT } |
   WITH COMPONENTS {..., objectDigestInfo  PRESENT } )

IssuerSerial ::= SEQUENCE {
  issuer     GeneralNames,
  serial     CertificateSerialNumber,
  issuerUID  UniqueIdentifier OPTIONAL,
  ... }

ObjectDigestInfo ::= SEQUENCE {
  digestedObjectType   ENUMERATED {
    publicKey        (0),
    publicKeyCert    (1),
    otherObjectTypes (2)},
  otherObjectTypeID   OBJECT IDENTIFIER OPTIONAL,
  digestAlgorithm     AlgorithmIdentifier{{SupportedAlgorithms}},
  objectDigest        BIT STRING,
  ... }

AttCertIssuer ::= [0]  SEQUENCE {
  issuerName               GeneralNames OPTIONAL,
  baseCertificateID  [0]  IssuerSerial OPTIONAL,
  objectDigestInfo   [1]  ObjectDigestInfo OPTIONAL,
  ... }
  (WITH COMPONENTS {..., issuerName  PRESENT } |
   WITH COMPONENTS {..., baseCertificateID  PRESENT } |
   WITH COMPONENTS {..., objectDigestInfo  PRESENT } )
```

```
AttCertValidityPeriod ::= SEQUENCE {
  notBeforeTime  GeneralizedTime,
  notAfterTime   GeneralizedTime,
  ... }

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
  otherName                   [0]  INSTANCE OF OTHER-NAME,
  rfc822Name                  [1]  IA5String,
  dNSName                     [2]  IA5String,
--x400Address                 [3]  ORAddress,
  directoryName               [4]  Name,
--ediPartyName                [5]  EDIPartyName,
  uniformResourceIdentifier   [6]  IA5String,
  iPAddress                   [7]  OCTET STRING,
  registeredID                [8]  OBJECT IDENTIFIER,
  ... }

OTHER-NAME ::= TYPE-IDENTIFIER

CertAVL ::= SIGNED {TBSCertAVL}

TBSCertAVL ::= SEQUENCE {
  version               [0]  IMPLICIT Version DEFAULT v1,
  serialNumber               AvlSerialNumber OPTIONAL,
  signature                  AlgorithmIdentifier {{SupportedAlgorithms}},
  issuer                     Name,
  constrained                BOOLEAN,
  entries                    SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    idType                     CHOICE {
      certIdentifier        [0]  PKCertIdentifier,
      entityGroup                DistinguishedName, -- only for constrained = FALSE
      ... },
    entryExtensions       [1]  IMPLICIT Extensions OPTIONAL,
    ... },
  ...,
  ...,
  avlExtensions              Extensions OPTIONAL }

AvlSerialNumber ::= INTEGER (0..MAX)

PKCertIdentifier ::= CHOICE {
  issuerSerialNumber         IssuerSerialNumber,
  fingerprintPKC        [0]  IMPLICIT FingerPrint {Certificate},
  fingerprintPK         [1]  IMPLICIT FingerPrint {PublicKey},
  ... }

IssuerSerialNumber ::= SEQUENCE {
  issuer        Name,
  serialNumber  CertificateSerialNumber,
  ... }

CRLReason ::= ENUMERATED {
  unspecified          (0),
  keyCompromise        (1),
  cACompromise         (2),
  affiliationChanged   (3),
  superseded           (4),
  cessationOfOperation (5),
  certificateHold      (6),
  removeFromCRL        (8),
  privilegeWithdrawn   (9),
  aACompromise         (10),
  ...,
  weakAlgorithmOrKey   (11) }

END
```

Bibliography

– Recommendation ITU-T X.207 (1993) | ISO/IEC 9545:1994, *Information Technology – Open Systems Interconnection – Application layer structure*.

– IETF RFC 2631 (1999), *Diffie-Hellman Key Agreement Method*.

– IETF RFC 5424 (2009), *The Syslog Protocol*.

– Fischlin M., Gunther, F., Schmidt, B., Warinschi, B. (2016). Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In: *2016 IEEE Symposium on Security and Privacy (SP*), San Jose, CA, USA, pp. 452-469. Piscataway, NJ: IEEE. doi: 10.1109/SP.2016.34

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks, open system communications and security**
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2020