



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**

CAUTION !

PREPUBLISHED RECOMMENDATION

This prepublication is an unedited version of a recently approved Recommendation. It will be replaced by the published version after editing. Therefore, there will be differences between this prepublication and the published version.

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/had not] 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/>.

© ITU 2020

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

**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 cyber security for protocols designed for protection by the wrapper protocol. The wrapper protocol provides authentication, integrity and optionally confidentiality (encryption). The wrapper protocol allows cyber security to be provided independently of the protected protocols, which means that the 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.

Keywords

Quantum-safe, cryptography; cryptographic algorithm; digital signature; public-key certificate; certification authority; distinguished name; PKI; trust anchor; validation.

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; and
- 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:

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.

CONTENTS

Page

SECTION 1 – GENERAL	7
1 Scope	7
2 Normative references	7
2.1 Identical Recommendations International Standards	7
2.2 Other references	7
3 Definitions	8
3.1 OSI Reference Model definitions	8
3.2 Directory model definitions	8
3.3 Public-key and attribute certificate definitions	8
3.4 Terms specified by this Recommendation International Standard	8
4 Abbreviations	9
5 Conventions	10
6 Common data types and special cryptographic algorithms	10
6.1 Introduction	10
6.2 ASN.1 information object class specification tool	10
6.2.1 General information object class concept	10
6.2.2 The ALGORITHM information object class	11
6.3 Multiple-cryptographic algorithm specifications	12
6.3.1 General	12
6.3.2 Multiple signatures algorithm	12
6.3.3 Multiple symmetric key algorithm	12
6.3.4 Multiple public-key algorithms	12
6.3.5 Multiple hash algorithm	13
6.3.6 Multiple authenticated encryption with associated data algorithm	13
6.3.7 Multiple integrity check value algorithm	13
6.4 Key establishment algorithms	13
6.4.1 General	13
6.4.2 Diffie-Hellman group 14 algorithm with HKDF-256	13
6.4.3 Diffie-Hellman group 23 algorithm with HKDF-256	14
6.4.4 Diffie-Hellman group 28 algorithm with HKDF-256	14
6.4.5 Key derivation	15
6.4.5.1 General	15
6.4.5.2 HMAC-based Extract-and-Expand Key Derivation Function	15
6.4.6 Special conditions	15
6.5 Multiple-cryptographic algorithm-value pairs	16
6.5.1 Multiple digital signatures attached to data	16
6.5.2 Double digital signature attached to data	16
6.5.3 Duplicate integrity check values attached to data	16
6.6 Formal specification of encipherment	17
6.6.1 Formal specification of encryption	17
6.6.2 Formal specification of authenticated encryption with associated data	17
7 General concepts for securing protocols	17
7.1 Introduction	17
7.2 Protected protocol plug-in concept	18
7.3 Communications structure	18
7.4 Another view of the relationship between the wrapper protocol and the protected protocol	19
7.5 Structure of application protocol data unit	19
7.6 Exception conditions	19
SECTION 2 – THE WRAPPER PROTOCOL	20
8 Wrapper protocol general concepts	20
8.1 Introduction	20

8.2	UTC time specification	20
8.3	Use of alternative cryptographic algorithms	20
8.4	General on establishing shared keys	20
8.5	Sequence numbers.....	21
8.6	Use of invocation identification in the wrapper protocol	21
8.7	Mapping to underlying services	21
8.8	Definition of protected protocols	21
8.9	Overview of wrapper protocol data units	21
9	Association management.....	22
9.1	Introduction to association management	22
9.2	Association handshake request.....	22
9.3	Association accept.....	24
9.4	Association reject due to security issues	25
9.5	Association reject by the protected protocol	26
9.6	Handshake security abort	27
9.7	Handshake abort by protected protocol.....	27
9.8	Data transfer security abort	28
9.9	Abort by protected protocol	28
9.10	Release request WrPDU.....	29
9.11	Release response WrPDU	29
9.12	Release collision.....	30
10	Data transfer phase	30
10.1	Symmetric keys renewal	30
10.2	Data transfer by the client	31
10.2.1	General.....	31
10.2.2	Client using authenticated encryption with associated data	31
10.2.3	Client not using authenticated encryption with associated data	31
10.2.4	Client non-encrypted data	32
10.3	Data transfer by the server	32
10.3.1	General.....	32
10.3.2	Server using authenticated encryption with associated data	32
10.3.3	Server not using authenticated encryption with associated data.....	33
10.3.4	Server non-encrypted data.....	33
11	Information flow.....	34
11.1	Purpose and general model	34
11.2	Protected protocol SAOC.....	35
11.3	Wrapper SAOC	35
11.3.1	General.....	35
11.3.2	Handshake request subclass	35
11.3.3	Handshake accept subclass	35
11.3.4	Handshake security reject subclass	36
11.3.5	Handshake reject by protected protocol subclass.....	36
11.3.6	Handshake security abort subclass.....	36
11.3.7	Handshake abort by protected protocol subclass.....	36
11.3.8	Data transfer security abort subclass.....	36
11.3.9	Data transfer application abort subclass.....	36
11.3.10	Release request subclass	37
11.3.11	Release response subclass.....	37
11.3.12	Client data transfer with authenticated encryption with associated data subclass.....	37
11.3.13	Client data transfer with integrity check value protection subclass	37
11.3.14	Server data transfer with authenticated encryption with associated data subclass	38
11.3.15	Client data transfer with integrity check value protection subclass	38
12	Wrapper error handling	38
12.1	General.....	38
12.2	Checking of a wrapper handshake request	38
12.2.1	General.....	38
12.2.2	Digital signature checking.....	38

12.2.3	Checking of the to-be-signed part	39
12.3	Checking of a wrapper handshake accept	39
12.3.1	General	39
12.3.2	Digital signature checking.....	39
12.3.3	Checking of the to-be-signed part	40
12.4	Checking of data transfer WrPDUs.....	41
12.4.1	General	41
12.4.2	Common checking for data transfer	41
12.4.2.1	Common checking for use of authenticate encryption with associated data	41
12.4.2.2	Common checking for non-use of authenticate encryption with associated data	41
12.4.2.3	Common checking for AadClient and -AadServer data types.....	41
12.4.5	AadClient data value specific checking	42
12.4.6	AadServer data value specific checking.....	42
12.5	Wrapper diagnostic codes	42
SECTION 3 – PROTECTED PROTOCOLS		44
13	Authorization and validation list management	44
13.1	General on authorization and validation management	44
13.1.1	Introduction	44
13.1.2	Invocation identification	44
13.1.3	Exception conditions	44
13.2	Defined protected protocol data unit (PrPDU) types.....	44
13.3	Authorization and validation management protocol initialization request.....	45
13.4	Authorization and validation management protocol initialization accept	45
13.5	Authorization and validation management protocol initialization reject.....	45
13.6	Authorization and validation management protocol initialization abort	45
13.7	Add authorization and validation list request	46
13.8	Add authorization and validation list response	47
13.9	Replace authorization and validation list request.....	47
13.10	Replace authorization and validation list response.....	47
13.11	Delete authorization and validation list request	48
13.12	Delete authorization and validation list response	48
13.13	Authorization and validation list abort.....	49
13.14	Authorization and validation list error codes	49
14	Certification authority subscription protocol.....	50
14.1	Certification authority subscription introduction	50
14.2	Defined protected protocol data unit (PrPDU) types.....	50
14.3	Certification authority subscription protocol initialization request	50
14.4	Certification authority subscription protocol initialization accept	51
14.5	Certification authority subscription protocol initialization reject.....	51
14.6	Certification authority subscription protocol initialization abort	51
14.7	Public-key certificate subscription request.....	52
14.8	Public-key certificate subscription response	52
14.9	Public-key certificate un-subscription request	53
14.10	Public-key certificate un-subscription response.....	54
14.11	Public-key certificate replacements request	55
14.12	Public-key certificate replacement response	55
14.13	End-entity public-key certificate updates request	56
14.14	End-entity public-key certificate updates response	56
14.15	Certification authority subscription abort.....	57
14.16	Certification authority subscription error codes.....	57
15	Trust broker protocol.....	58
15.1	Introduction.....	58
15.2	Defined protected protocol data unit (PrPDU) types.....	58
15.3	Trust broker protocol initialization request	59
15.4	Trust broker protocol initialization accept	59

15.5	Trust broker protocol initialization reject.....	59
15.6	Trust broker protocol initialization abort	59
15.7	Trust broker request syntax.....	60
15.8	Trust broker response syntax.....	60
15.9	Trust broker error information	60
Annex A	Crypto Tools in ASN.1.....	62
Annex B	Wrapper protocol in ASN.1.....	65
Annex C	Protected protocol interface to the wrapper protocol.....	70
Annex D	Cryptographic algorithms.....	72
Annex E	Authorization and validation list management in ASN.1	74
Annex F	Certification authority subscription in ASN.1.....	77
Annex G	Trust broker in ASN.1.....	81
Annex H	Migration of cryptographic algorithms	83
H.1	Introduction.....	83
H.2	Negotiation of cryptographic algorithms	83
H.2.1	Cryptographic negotiation for new protocols.....	83
H.2.2	Cryptographic negotiation for existing protocols.....	83
H.3	Non-negotiable digital signature algorithms	84
H.3.1	General.....	84
H.3.2	Duplicate signatures for new protocols	85
H.3.3	Duplicate signatures for existing protocols.....	85
Annex I	Auxiliary specifications	87
Bibliography		92

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:

It provides guidance for how to prepare new and old protocols for cryptographic algorithm migration. It defines auxiliary cryptographic algorithms to be used for migration purposes

It is within the scope of this Recommendation | International Standard to provide 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.

The scope 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.509 (2019) | ISO/IEC 9594-8:2020, *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks*.
- 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.660 (2011) | ISO/IEC 9834-1:2012, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: General procedures and top arcs of the International Object Identifier tree*.
- 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.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 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.200 | ISO 7498-1:

- a) abstract syntax;
- b) confidentiality;
- c) cryptography; and
- d) digital signature.

3.2 Directory model definitions

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

- a) attribute; and
- b) distinguished name.

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; and
- o) trust broker.

3.4 Terms specified by this Recommendation | International Standard

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

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

- 3.4.3 application protocol data unit (APDU):** Data that is transmitted as a single unit at the application layer between two application entities.
- 3.4.4 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.
- 3.4.5 client:** The entity that initiates an association.
- 3.4.6 data transfer phase:** The phase from the completion of the establishment of an association to the termination of the association.
- 3.4.7 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.
- 3.4.8 native cryptographic algorithm:** A cryptographic algorithm used prior to a migration period.
- 3.4.9 protected protocol data unit (PrPDU):** Application protocol data unit (APDU) defined by an application protocol to be protected by the wrapper protocol.
- 3.4.10 server:** The entity that accept or reject an association.
- 3.4.11 symmetric key:** A cryptographic key used for both encryption of plaintext and decryption of ciphertext.
- 3.4.12 wrapper protocol data unit (WrPDU):** An application protocol data unit (APDU) 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
APDU	Application Protocol Data Unit
ASN.1	Abstract Syntax Notation One
AVL	Authorization and Validation List
AVMP	Authorization 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
IANA	Internet Assigned Numbers Authority
ICV	Integrity Check Value
ICT	Information and Communications Technology
IETF	Internet Engineering Task Force
MAC	Message Authentication Code
PDU	Protocol Data Unit
PKI	Public-Key Infrastructure
PMI	Privilege Management Infrastructure
PrPDU	Protected protocol Data Unit
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 Rec. ITU-T X.510 | ISO/IEC 9594-11.

The term "The Directory Specifications" shall be taken to mean Rec. ITU-T X.500 | ISO/IEC 9594-1, Rec. ITU-T X.501 | ISO/IEC 9594-2, Rec. ITU-T X.511 | ISO/IEC 9594-3, Rec. ITU-T X.518 | ISO/IEC 9594-4, Rec. ITU-T X.519 | ISO/IEC 9594-5, Rec. ITU-T X.520 | ISO/IEC 9594-6, Rec. ITU-T X.521 | ISO/IEC 9594-7 and Rec. ITU-T X.525 | ISO/IEC 9594-9.

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:

- ASN.1 information object classes are heavily used for protocol design. The **ALGORITHM** information object class is important for this Specification. This further expanded in clause 6.2.
- 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 below.
- It has advantages to define cryptographic algorithms for key agreement procedures. This has been done for a few cases in clause 6.4.
- Parameterized data types are defined in clause 6.5 for flexible protocol design.
- Some formal specifications for encipherment are defined in clause 6.6.

6.2 ASN.1 information object class specification tool

6.2.1 General information object class concept

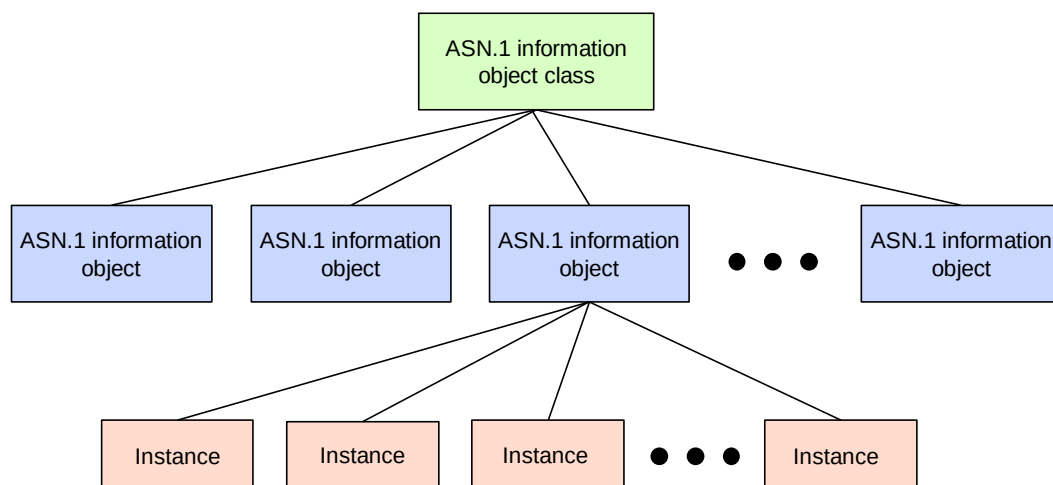


Figure 1 – ASN.1 information object class concept

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 inspire increase 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 and/or may be stored in a directory.

NOTE – The concept of object classes used in clause 11 is somewhat different that the concept of information object class as 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 for defining 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 copied below 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 for specifying 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:

- The **&Type** field is used for specifying 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.
- The **&DynParms** field is used for specifying 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.
- The **&id** field is used for uniquely identify the class of cryptographic algorithm being defined.

The **AlgorithmWithInvoke** parameterized data type defined below 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;

- The **algorithm** component shall hold the object identifier that uniquely identify the cryptographic algorithm being defined.
- 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.
- 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 below 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 as for the corresponding components of the **AlgorithmWithInvoke** parameterized data type.

The **AlgoInvoke** parameterized data type defined below 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 is 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 for establishing symmetric keys directly between two communicating entities in a secure way. The Diffie-Hellman technology, as specified in IETF RFC 2631, is an important example. The result of a Diffie-Hellman operation is a shared secret that a key derivation technology expands to generate the required symmetric keys.

Diffie-Hellman and the key derivation technologies in their basic form are not well suited for the cryptographic algorithm migration techniques as described in Annex H. By defining different combination of Diffie-Hellman 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.

This clause is provided for the purpose of validating the wrapper protocol as a whole using Diffie-Hellman 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 the 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 Diffie-Hellman key agreement technic.

```
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 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 Diffie-Hellman public key to be used by the sender. For the used group, the length of the key *g* is always 256 octets. A different Diffie-Hellman private key/public key pair shall be generated for each new association establishment. For the duration of the association, the Diffie-Hellman public key of the server shall be retained by both parties and the server shall retain its Diffie-Hellman private key for later renewal of symmetric keys (see clause 10.2).
- b) For each key renewal, the client shall generate a new Diffie-Hellman private/ Diffie-Hellman public key pair and use this pair for the key renewal process.
- c) The **nonce** component shall hold a random value to be used by the shared keys derivation specified in clause 6.4.5. A new value shall be generated for each key establishment. The length is recommended to be length of the hash output. As the **sha256** algorithm 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 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 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 by use of a key derivation function be expanded to provide sufficient material for the required symmetric keys.

6.4.5.2 HMAC-based Extract-and-Expand Key Derivation Function

The HMAC-based Extract-and-Expand Key Derivation Function (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:

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 Diffie-Hellman key agreement.
- c) The **OKM** is defined as:

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

- **PRK** shall be the **PRK** generated by the **HKDF-extract** 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; and
- 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 for generating an ICV by the client;
- the second key is the key used for generating an ICV by the server;
- the third key is the key used for encryption by the client; and
- 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}),
    parameters     SEQUENCE SIZE (1..MAX) OF
        sign      SEQUENCE {
            algo      AlgorithmIdentifier({SupportedSignatureAlgorithms}),
            signature  BIT STRING,
            ... },
    ... }
```

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

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:

- The **toBeSigned** component shall hold the value of the data type to be signed.
- 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 signaled.
- The **altSignature** component may be present if an alternative digital signature algorithm previously has been signaled. 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 to be integrity 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:

- The **toBeProtected** component shall the data value of what is passed in parameter being the data value to be protected.
- The **algorithmIdentifier** component consists of the following component (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 ICV algorithm identified by the **algorithm** and **parameters** components above 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 the data value of what is passed in 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 for specifying AEAD.

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

IETF RFC 5084 is a specification on the use of authenticated encryption with associated data. This RFC has associated data as optional, while it is mandatory by this Specification.

7 General concepts for securing protocols

7.1 Introduction

Protocols used in ICT infrastructures need to include cyber security capabilities to make them resistant to attacks. The intention of this Specification is to provide a structure where the cyber security aspects are clearly separated from the

actual protocol requiring cyber security. This is done by a specification of general protocol, called the wrapper protocol, which includes all the cyber security aspects. This wrapper protocol can then imbed another protocol, called the protected protocol, and in this way supply cyber security to the imbedded 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 and/or a PMI, but it may also be used for protocols with different purposes.

7.2 Protected protocol plug-in concept

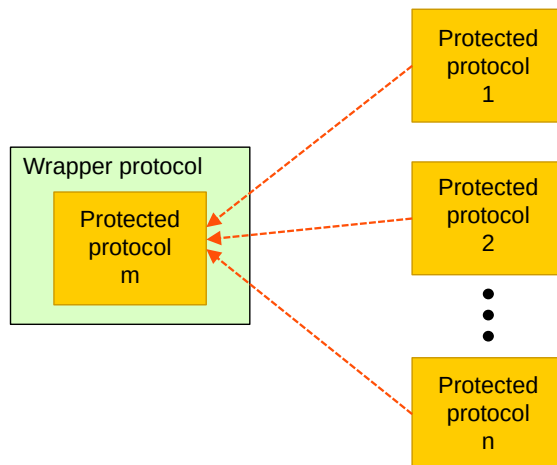


Figure 2 – Protected protocol plug-in

Figure 2 is one way of illustrating the relationship between the wrapper protocol and a protected protocol. A protected protocol is seen as imbedded 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).

7.3 Communications structure

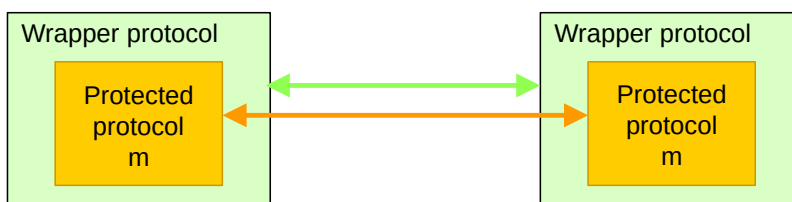


Figure 3 – Imbedded communication

Figure 3 depicts the communications between two entities 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 protocols may also need to communicate without involving the instances of the protected protocol.

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

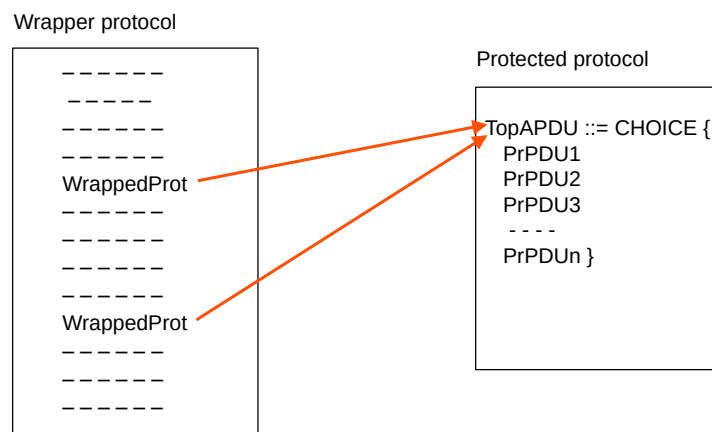


Figure 4 – Inclusion of the protected protocol

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

7.5 Structure of application protocol data unit

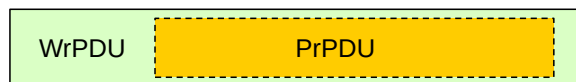


Figure 5 – APDU structure

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 imbedded in the WrPDU in the form of a protected application protocol data unit (PrPDU). When the two instances of the wrapper protocol need to interact without involving the protected protocol, there will be no imbedded PrPDU.

7.6 Exception conditions

This Specification dictates 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 exceptions:

- 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.
- Errors that may safely assumed not being 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:

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

The purpose of Section 2 is to specify the general wrapper protocol providing security to other protocols. Clause 8 (this clause) makes some general consideration general for the wrapper protocol. Clause 9 specifies the WrPDUs used for establishment, termination and abortion of associations. Clause 10 specifies the WrPDUs used during the data transfer phase. Clause 11 gives an analysis of 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 relates 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 for their creation. The time is given as 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 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 24-hour clock system, mm is the minute and the ss is the seconds. A comma (',') or full stop ('.') is used as the decimal mark.
- c) The accuracy shall be either one hour, one minute, one second 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 the document. The cryptographic mechanisms and algorithms used by the end-users of the wrapper protocol shall comply with its national legislation.

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

8.4 General on establishing shared keys

To encrypt a wrapped PrPDU and/or generating an integrity check value (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; and
- 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 Diffie-Hellman (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 detecting of replay of WrPDUs caused by an error or by an adversary; and
- to detect missing WrPDUs.

The sequence number is 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 six-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 character are numeric characters take 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 onto 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:

- The **id** component shall hold the object identifier assigned to the protected protocol in question.
- 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 in clause 7.4).

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 during a migration period contain two digital signatures. 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 – Currently, 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 for generating 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 purpose (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 for generating 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 for validating 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 DER encoded allowing other than 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 of this component was created (see clause 8.2 for details).
- h) The **keyEst** component shall hold the key establishment algorithm used for establishing shared symmetric keys to be used during data transfer. Two or four keys shall be generated as specified in item j) below.
- 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:
 - 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 algorithms it supports of the suggested AEAD algorithms.

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

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

This alternative has two components:

- 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 algorithms it supports of the suggested algorithms.
 - 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 algorithms it supports of the suggested algorithms.
- 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 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 what digital signature algorithm shall be used in future communications within the association.

The selected digital signature algorithm shall be used for generating 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 for verifying 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 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-algorithms 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 the symmetric key algorithms suggested by the client.
 - ii) The **icvAlgID** component shall select the first supported ICV algorithm of the sequence-of of the ICV algorithms 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 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 algorithm, it shall instead add a supported algorithm in the **sigAlg** alternative.

The selected digital signature algorithm shall be used for generating 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 for verifying 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 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 for generating 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 for verifying 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 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 for generating 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 for verifying 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.