International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.1080.0
Corrigendum 1

(03/2018)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

Information and network security – Telebiometrics

Access control for telebiometrics data protection

Corrigendum 1

Recommendation ITU-T X.1080.0 (2017) – Corrigendum 1

ITU-T

International Telecommunication Union

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	X.1–X.199
OPEN SYSTEMS INTERCONNECTION	X.200–X.299
INTERWORKING BETWEEN NETWORKS	X.300–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	X.600–X.699
OSI MANAGEMENT	X.700–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	X.850–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	
General security aspects	X.1000–X.1029
Network security	X.1030–X.1049
Security management	X.1050–X.1069
Telebiometrics	**X.1080–X.1099**
SECURE APPLICATIONS AND SERVICES (1)	
Multicast security	X.1100–X.1109
Home network security	X.1110–X.1119
Mobile security	X.1120–X.1139
Web security	X.1140–X.1149
Security protocols (1)	X.1150–X.1159
Peer-to-peer security	X.1160–X.1169
Networked ID security	X.1170–X.1179
IPTV security	X.1180–X.1199
CYBERSPACE SECURITY	
Cybersecurity	X.1200–X.1229
Countering spam	X.1230–X.1249
Identity management	X.1250–X.1279
SECURE APPLICATIONS AND SERVICES (2)	
Emergency communications	X.1300–X.1309
Ubiquitous sensor network security	X.1310–X.1319
Smart grid security	X.1330–X.1339
Certified mail	X.1340–X.1349
Internet of things (IoT) security	X.1360–X.1369
Intelligent transportation system (ITS) security	X.1370–X.1389
Distributed ledger technology security	X.1400–X.1429
Security protocols (2)	X.1450–X.1459
CYBERSECURITY INFORMATION EXCHANGE	
Overview of cybersecurity	X.1500–X.1519
Vulnerability/state exchange	X.1520–X.1539
Event/incident/heuristics exchange	X.1540–X.1549
Exchange of policies	X.1550–X.1559
Heuristics and information request	X.1560–X.1569
Identification and discovery	X.1570–X.1579
Assured exchange	X.1580–X.1589
CLOUD COMPUTING SECURITY	
Overview of cloud computing security	X.1600–X.1601
Cloud computing security design	X.1602–X.1639
Cloud computing security best practices and guidelines	X.1640–X.1659
Cloud computing security implementation	X.1660–X.1679
Other cloud computing security	X.1680–X.1699

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

Recommendation ITU-T X.1080.0

Access control for telebiometrics data protection

Corrigendum 1

Summary

Recommendation ITU-T X.1080.0 provides specifications on how to protect telebiometrics information against unauthorized access. A service-oriented view is taken, where only information necessary for a particular purpose is provided, i.e., access is given not only on a *right-to-know* basis, but also on a *need-to-know* basis.

The core of this Recommendation is an attribute specification included in an attribute certificate or public-key certificate that specifies in detail what privileges a particular entity has for one or more service types.

Security is provided by using a profile of the cryptographic message syntax (CMS). The CMS profile provides authentication, integrity and, when required, confidentiality (encryption).

This profile is intended to provide security support for telebiometrics specifications in general. The profile assumes, and is dependent upon, the correct deployment of a public-key infrastructure (PKI).

This Recommendation is also dependent on the deployment of a privilege management infrastructure (PMI).

Corrigendum 1 corrects minor editorial issues in Annex A as well as some errors in the object identifier allocations in Annex A. The corrections to Annex A are also applied in Annex C and Appendix I.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T X.1080.0	2017-03-30	17	11.1002/1000/13193
1.1	ITU-T X.1080.0 (2017) Cor. 1	2018-03-29	17	11.1002/1000/13591

Keywords

Access control, Diffie-Hellman, PKI, telebiometrics.

---

* 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 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/.

Table of Contents

Introduction

When collecting telebiometrics data from individuals, there is a risk of infringement of privacy.

There may be several reasons for protecting such information. Information may provide access to a company or an organization or it may be of a sensitive nature that restricts its distribution.

The protection against unwanted disclosure of telebiometrics data has two major aspects:

– protection of data during transmission, typically by encryption, and protection of stored data;

– control of access to stored data.

While telebiometric systems should have a high level of security with respect to confidentiality (encryption), authentication, integrity, physical protection, use of firewalls, virus protection programs, etc., it is also necessary to establish an elaborate access control system for access to stored information, in particular information about individuals. This latter aspect is particularly important for telebiometric systems.

General access control schemes have a shortcoming in that they mainly consider the *right* (or refusal) to information, but do not consider, in detail, the *need-to-know* aspect. A need-to-know implies that it is not sufficient to have the right to see the data, but it also must be established that this information is to be used for legitimate purposes.

Information should only be provided for its intended use. Medical information about a patient is collected to allow optimal treatment of that patient and should not be used for any other purpose, except possibly for tightly controlled research projects that may require use of some pieces of information from a specific collection of patients, otherwise information shall be protected against information trawling.

There are two main types of access control: physical and logical. Physical access control limits access to campuses, buildings, rooms and physical information technology (IT) assets. Logical access limits connections to computer networks, system files and data. Only logical access control is considered in this Recommendation

Access control includes secure authentication of accessors by the service provider. This Recommendation assumes the use of digital signatures and an established public-key infrastructure (PKI).

Recommendation ITU-T X.1080.0 may be referenced by other telebiometrics specifications.

Annex A, which is an integral part of this Recommendation, specifies the allocation of object identifiers used by the ITU-T X.1080-series of Recommendations.

Annex B, which is an integral part of this Recommendation, provides a telebiometrics profile of the cryptographic message syntax (CMS), as discussed in IETF RFC 5652, that is used by this Recommendation. It can also be referenced by other telebiometrics specifications.

Annex C, which is an integral part of this Recommendation, provides a formal specification for the privilege assertion and assignment protocols in the form an Abstract Syntax Notation One (ASN.1) module.

Appendix I, which is not an integral part of this Recommendation, provides an informal specification for the CMS profile in the form of an ASN.1 module.

Recommendation ITU-T X.1080.0

Access control for telebiometrics data protection

Corrigendum 1

Editorial note: This is a complete-text publication. Modifications introduced by this corrigendum are shown in revision marks relative to Recommendation ITU-T X.1080.0 (2017).

1 Scope

This Recommendation provides a specification for how privacy may be protected in a telebiometrics environment by using privacy-based access control for telebiometrics (ACT). While this Recommendation cannot identify all possible information types, it is within the scope to provide general tools for handling all types of information in a secure way. This includes definition of a protocol for assigning privileges and a protocol for accessing information using privilege assertion. This Recommendation provides guidelines and does not contain compliance requirements.

The following is outside the scope of this Recommendation:

– physical protection of the information;

– unauthorized access by operational personal that maintain the security system and therefore having the possibility to circumvent security.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T X.500-series] Recommendation ITU-T X.5xx (2016) | ISO/IEC 9594-x series, *Information technology – Open Systems Interconnection – The Directory*.

[ITU-T X.501] Recommendation ITU-T X.501 (2016) | ISO/IEC 9594-2, *Information technology – Open Systems Interconnection – The Directory: Models*.

[ITU-T X.509] Recommendation ITU-T X.509 (2016) | ISO/IEC 9594-8, *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks*.

[ITU-T X.520] Recommendation ITU-T X.520 (2016) | ISO/IEC 9594-6, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types*.

[ITU-T X.521] Recommendation ITU-T X.521 (2016) | ISO/IEC 9594-7, *Information technology – Open Systems Interconnection – The Directory: Selected object classes*.

[ITU-T X.680] Recommendation ITU-T X.680 (2015) | ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.

[ITU-T X.681] Recommendation ITU-T X.681 (2015) | ISO/IEC 8824-2, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.

[ITU-T X.682] Recommendation ITU-T X.682 (2015) | ISO/IEC 8824-3, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.

[ITU-T X.683] Recommendation ITU-T X.683 (2015) | ISO/IEC 8824-4, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.

[ITU-T X.690] Recommendation ITU-T X.690 (2015) | ISO/IEC 8825-1, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*

[ITU-T X.1080.1] Recommendation ITU-T X.1080.1 (2011), *e-Health and world-wide telemedicines – Generic telecommunication protocol*.

[ITU-T X.1081] Recommendation ITU-T X.1081 (2011), *The telebiometric multimodal model – A framework for the specification of security and safety aspects of telebiometrics*.

[IETF RFC 2631] IETF RFC 2631 (1999), *Diffie-Hellman Key Agreement Method*.

[IETF RFC 3185] IETF RFC 3185 (2001), *Reuse of CMS Content Encryption Keys*.

[IETF RFC 5083] IETF RFC 5083 (2007), *Cryptographic Message Syntax (CMS) – Authenticated-Enveloped-Data Content Type*.

[IETF RFC 5652] IETF RFC 5652 (2009), *Cryptographic Message Syntax (CMS).*

[IETF RFC 5753] IETF RFC 5753 (2010), *Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax CMS*.

[IETF RFC 5911] IETF RFC 5911 (2010), *New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME*.

[IETF RFC 6268] IETF RFC 6268 (2011), *Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX).*

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 attribute certificate [ITU-T X.509]: A data structure, digitally signed by an attribute authority that binds some attribute values with identification information about its holder.

3.1.2 attribute type [ITU-T X.501]: That component of an attribute which indicates the class of information held by that attribute.

3.1.3 attribute value [ITU-T X.501]: A particular instance of the class of information indicated by an attribute type.

3.1.4 privilege [ITU-T X.509]: An attribute or property assigned to an entity by an authority.

3.1.5 privilege holder [ITU-T X.509]: An entity that has been assigned privilege. A privilege holder may assert its privilege for a particular purpose.

3.1.6 privilege verifier [ITU-T X.509]: An entity verifying certificates against a privilege policy.

3.1.7 source of authority (SOA) [ITU-T X.509]: An attribute authority that a privilege verifier for a particular resource trusts as the ultimate authority to assign a set of privileges for asserting that resource.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 access control: A security technique used to regulate who can do what to information resources in a computing environment.

3.2.2 access service: A service provided by a service provider for the execution of a particular transaction.

3.2.3 accessor: A privilege holder that accesses a particular access service using its privilege.

3.2.4 attribute: A piece of information of a particular type that is associated with an object. Information associated with an object is composed of attributes.

3.2.5 data protection domain: A domain where the information to be protected is under a single management component.

3.2.6 distinguished name: A name identifying an object that is unique within a specific context and which is formed by one or more name components reflecting the object's position in a hierarchy of objects.

3.2.7 object: A person, a department, a professional or some other type of object about which there is information and which is identifiable by a distinguished name.

3.2.8 object class: An identified family of entities that share certain characteristics.

3.2.9 operation: An interaction comprised of a request and a reply between an accessor and a service provider for a particular objective.

3.2.10 specification: An ITU-T Recommendation, an International Standard or any specification developed by a recognized Standards Developing Organization (SDO).

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

AA	Attribute Authority
ABAC	Attribute-based Access Control
ACL	Access Control List
ACT	Access Control for Telebiometrics
AES	Advanced Encryption Standard
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CA	Certification Authority
CEK	Content Encryption Key
CMS	Cryptographic Message Syntax
DER	Distinguished Encoding Rules
DH	Diffie-Hellman

ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
GCM	Galois/Counter Mode
KEK	Key-Encryption Key
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
PDU	Protocol Data Unit
PKI	Public-Key Infrastructure
PMI	Privilege Management Infrastructure
SDO	Standards Developing Organization
SOA	Source of Authority

5 Conventions

This Recommendation presents Abstract Syntax Notation One (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.

If the items in a list are numbered (as opposed to using "–" or letters), then the items shall be considered steps in a procedure.

6 Basic concepts and models

6.1 Protection in a single data protection domain

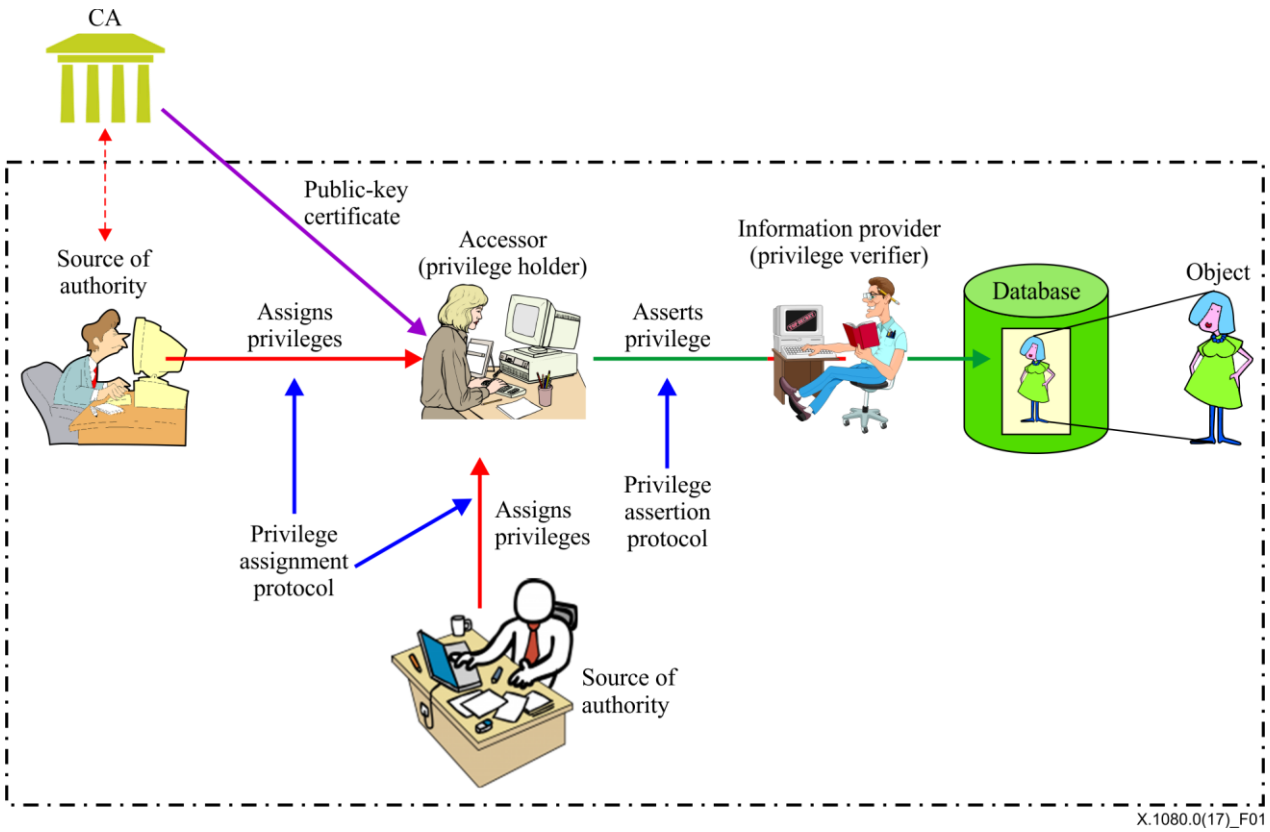

Figure 1 – Single data protection domain model

Figure 1 illustrates the various partners and protocols in an access control environment within a single data protection domain. A data protection domain comprises all entities involved in protection that are under common management.

An entity, called an accessor, may have a job function that requires permission to perform operations on information about objects held by an information provider. As an example, a doctor who is responsible for a patient needs to access the patient's health record to provide optimal treatment, while other doctors not involved with the patient have no need for such information.

Permission to perform a particular operation on information should only be given if the accessor has the *right* and a *genuine need* to perform that operation, i.e., it has been assigned the necessary *privilege*.

It outside the scope of this Recommendation to specify under what conditions privileges are assigned to entities. It is only within the scope to provide the necessary tools for managing privileges in a secure way.

A data protection domain needs to establish one or more authorities to assign privileges to entities. The term source of authority (SOA) defined in [ITU-T X.509] is used here as the term for the ultimate authority for assigning privileges for a specific area within a data protection domain.

This Recommendation uses certificates to assign privilege to accessors. Privilege may be carried in in attribute certificates in the **`attributes`** component or in public-key certificates in the **`subjectDirectoryAttributes`** extension. Privileges that are permanent may typically be provided in public-key certificates, while temporary privileges may typically be provided in attribute certificates.

Figure 1 illustrates the relationship among the various components within a single data protection domain. It shows two SOAs each responsible for assigning privileges for different aspects to privilege holders.

Some privileges may be needed for daily operations of the accessor and are therefore of a more permanent nature, while other privileges are assigned for handling special situations and are therefore temporary in nature.

When privileges are assigned to an accessor, the assessor becomes a privilege holder and may use their privilege to access information through the privilege assertion protocol. The privilege verifier, representing the information provider, checks the asserted privilege before allowing access to the information.

An SOA may forward privileges either by local means or by embedding them in a signed attribute certificate in the privilege assignment protocol as illustrated in Figure 1.

6.2 Cross data protection domain

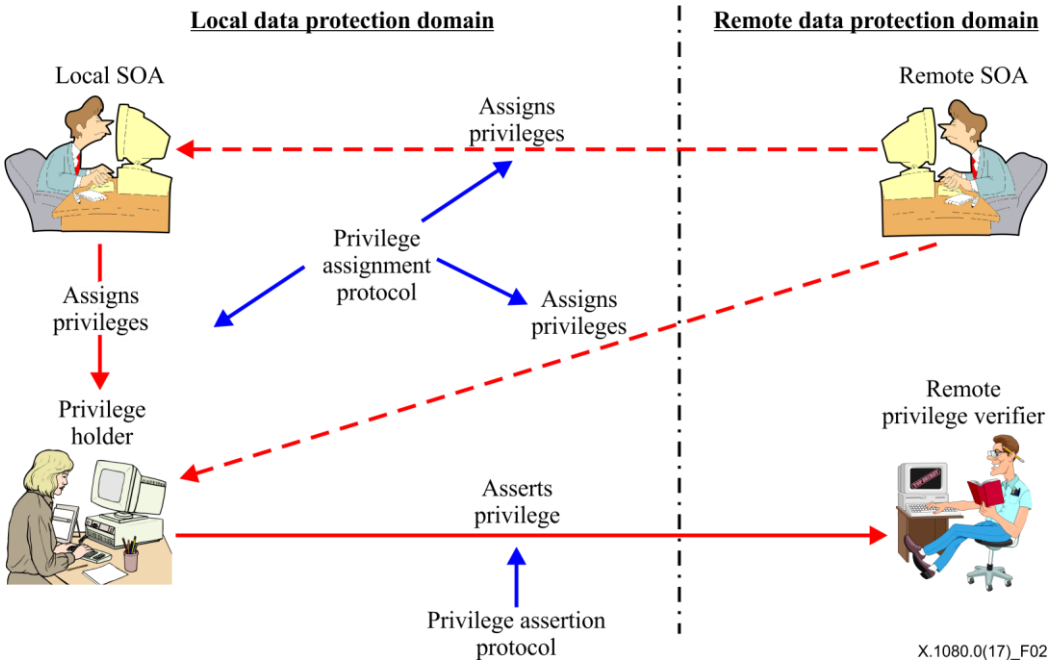

Figure 2 – Cross data protection domain model

Figure 2 illustrates the case where an accessor needs access to information within another data protection domain, e.g., to treat a patient for which vital health information is located in another data protection domain.

In this environment, the accessor either directly or through a local SOA requests access to specific information about a particular object. How this request is forwarded is outside the scope of this Recommendation, but it could be by e-mail or by telephone.

The remote SOA generates an attribute certificate with an attribute of type `accessService` holding the necessary privilege. This attribute certificate is signed by the remote SOA. The `holder` of the attribute certificate may be either the local SOA or it may be the entity that needs the privilege.

If it is the local SOA having the privilege assigned, this SOA:

a) decrypts the content if required;

b) verifies the signature on the content;

c) extracts the attribute certificate from the content;

d) validates extracted attribute certificate;

e) creates a new attribute certificate with the privilege information taken from the received attribute certificate and with the entity needing the privilege as the holder;

f) this attribute certificate together with the one received from the remote SOA is then forwarded to the entity needing the privilege (i.e., the privilege holder).

6.3 Service-oriented model

The access control for telebiometrics is service oriented in the sense that an accessor may invoke a set of distinct functions, called *access services*. An access service is a function provided by a service provider capable of performing the particular access service. An example of an access service is the capability to manipulate patient information within a hospital. An object identifier is used to identify a particular access service type. This Recommendation does not define specific access services, but

provides the tools for establishing access services. Referencing specifications may define access services relevant for their scope.

Although an accessor has access to a particular service, they may not have the privilege that covers all aspects of that service.

6.4 The object and attribute model

The information model defined in [ITU-T X.501] is designed to handle different data structures. This Recommendation uses this information model for access control.

In the ITU-T X.501 model, objects to be protected are organized in object classes, where objects within an object class have common characteristics relevant within a specific context. An object class is identified by an object identifier. [ITU-T 1080.1] expands on this concept by assigning object identifiers to a group of object classes for categories relevant to telebiometrics.

[ITU-T X.521] also defines a set of general usable object classes. In cases where an object class that is not directly related to telebiometrics is required, an already defined object class should be used whenever possible.

The individual objects are identified by distinguished names as defined in [ITU-T X.501]. A distinguished name consists of one or more name components that reflect its position within an object hierarchy.

The information associated with an object is modelled as a set of attributes. Attributes with common characteristics constitute an attribute type. An attribute type is identified by an object identifier. An attribute is a sequence of an object identifier identifying the attribute type and one or more values of that type. An object may only have one attribute of a particular type. [ITU-T X.520] defines a set of general useable attribute types. Already defined attribute types should be used whenever possible. Additional attribute types may be defined as required by individual specifications.

Individual usages of this Recommendation will have to provide a mapping between this information model and the actual data base structure. This mapping becomes easy if the information is maintained in a lightweight directory access protocol (LDAP) directory or in a secure directory based on the specifications in the [ITU-T X.500-series].

6.5 Basic access control principles

The purpose of this clause is to provide an overview of the general principles for establishing access control for telebiometrics. Traditional access control is mostly concerned with the right or refusal of access to data, based on some permanent parameters. This Recommendation takes an extended approach by also considering the need-to-know aspects. This is done by taking the service approach as discussed in clause 6.3. To perform a particular operation, an accessor needs to have privilege permitting access to the relevant access service and privilege permitting access to the necessary information.

This privilege includes access to objects of one or more object classes, possibly limited to some named objects. The type of operation that may be performed, may be different for different object classes and named objects.

The operations on objects may include operation on individual attributes and attribute values. The permitted operations may be different for different types of attributes.

6.6 Relationship to other access control schemes

There are various types of access controls to cover different access control requirements. It is the intension here to briefly describe various access control schemes and to relate them to ACT.

6.6.1 Basic access control as defined by ITU-T X.501

The basic access control as defined by [ITU-T X.501] is used for protecting information in a directory as defined by [ITU-T X.500-series]. It provides for elaborate access control lists that specifies, in detail, how and what different users are allowed to access. It differs from this Recommendation by providing only for the right-to-know, but not for the need-to-know.

6.6.2 Rule-based access control

Both [ITU-T X.501] and [b-ITU-T X.841] define a type of rule-based access control. In this type of access control, the data to be protected are tagged with information about what protection is required, while the accessing users have certified information associated in an access request that specifies the level of clearance they have with respect to accessing certain information. This concept differs from this Recommendation by requiring tagging of stored data items, and by only providing for right-to-know, but not for need-to-know.

6.6.3 Role-based access control as defined for smart grid

Role-based access control as specified in [b-IEC 62351-8] focuses on users and the jobs users perform. A role is a collection of rights to objects (actions that can be performed at certain targets). A user may have one or several roles. For access control, a role is a kind of intermediary that reduces the amount of information needed in the access control list (ACL) by decreasing the granularity of access control information. Permissions to objects of the system are not listed for each user separately, but users are given roles, and rights of each role are only described once.

6.6.4 Attribute-based access control

Attribute-based access control (ABAC) is a logical access control model that is distinguishable because it controls access to objects by evaluating rules against the attributes of entities (subject and object), operations, and the environment relevant to a request. ABAC systems are capable of enforcing both discretionary access control and mandatory access control concepts. ABAC enables precise access control, which allows for a higher number of discrete inputs into an access control decision, providing a greater set of possible combinations of those variables to reflect a larger and more definitive set of possible rules to express policies.

For a good introduction to ABAC, see [b-NIST 800-162].

6.7 Protocols overview

6.7.1 Privilege assessment protocol

The privilege assessment protocol comprises a set of cryptographic message syntax (CMS) content types. Each type of access requires a request content type and a result content type.

An instance of a content type is transmitted using CMS as profiled in Annex B. A formal ASN.1 module is provided in Annex C.

6.7.2 Privilege assignment protocol

The privilege assignment protocol is used for transmitting attribute certificates either between SOAs or from an SOA to the privilege holder.

A single pair of content types is defined for this protocol: one content type for forwarding privilege in the form of attribute certificates, and one content type for reporting the outcome.

An instance of a content type is transmitted using CMS as profiled in Annex B. A formal ASN.1 module is provided in in Annex C.

6.8 Use of CMS

A profile for telebiometrics use of CMS is provided in Annex B.

To ensure that the source of information is properly authenticated, contents of types defined by this Recommendation shall be encapsulated in an instance of the **signedData** content type. As sensitive information may be transmitted, it is recommended also to encapsulate it in an instance of the **envelopedData** content type. The ct-**autEnvelopedData** content type is not used by this Recommendation.

6.9 Public-key certificate considerations

An attribute certificate shall be signed by the issuer using its private key and it is to be verified by the use of the corresponding public-key certificate issued to the attribute certificate issuer.

In principle, the same private key could be used for signing the CMS message using the **signedData** content type, which would ease the verification process. However, there are security concerns using the same private key for different purposes. The use of different private keys for the two purposes should be considered, but is not mandated by this Recommendation.

7 Provision of privilege information

7.1 Use of attribute certificates

[ITU-T X.509] allows both public-key certificates and attribute certificates to carry privilege information. In both cases, the privilege specifications are carried in attributes as defined by [ITU-T X.501]. Attribute types for this purpose are distinct from attribute types used for modelling information about objects. While attribute types defined for carrying information about objects should be kept as simple as possible, attribute types for carrying privilege information will, by their nature, be rather complex.

When using an attribute certificate to carry privilege information:

a) the **holder** component identifies the entity to which privileges are to be assigned. When a privilege holder presents this attribute, certificate holding privileges to a privilege verifier, the privilege verifier shall authenticate the accessor to verify that the accessor is actually the privilege holder of the attribute certificate;

b) the **issuer** component shall hold the name of the SOA or the name of an attribute authority (AA) that has been delegated to assign privileges. The privilege verifier shall also obtain and validate the public-key certificate of the issuer to validate the signature on the attribute certificate;

c) the **attributes** component shall hold an attribute of the type **accessService**, as defined in clause 7.3.

The attribute certificate shall be signed by the SOA that approved the privilege or the AA to which issuing has been delegated.

Validation will be simplified if the holder and the issuer have public-key certificates issued by the same certification authority (CA).

7.2 Use of public-key certificates

When using a public-key certificate to carry privilege information:

a) the **subject** component identifies the accessor to which privileges have been assigned. When an accessor presents this public-key certificate holding privileges to a privilege verifier, the privilege verifier shall authenticate the accessor;

b) the **issuer** component shall hold the distinguished name of the CA that is responsible for issuing the public-key certificate;

c) the **subjectDirectoryAttributes** extension shall hold an instance of the **accessService** attribute type (see clause 7.3).

7.3 The access service attribute type

7.3.1 Access service attribute syntax

An attribute of type **accessService** is intended to be included in the **attributes** component of an attribute certificate or in the **subjectDirectoryAttributes** extension of a public-key certificate. It has the following syntax:

```
AccessService ATTRIBUTE ::= {
  WITH SYNTAX  AccessService
  ID           id-at-accessService }
```

An attribute of type **AccessService** provides privilege information to allow a privilege verifier to validate whether an access request should be honoured or not. It is a multi-valued attribute type allowing multiple access service types and associate permissions to be included in the same attribute.

An entity cannot use a service that is not included in this attribute.

The **accessService** attribute type has the following syntax:

```
AccessService ::= SEQUENCE {
  serviceId          OBJECT IDENTIFIER,
  objectDef          SEQUENCE SIZE (1..MAX) OF ObjectSel,
  ... }
```

The components of a value of the **AccessService** data type are:

a) the **serviceId** component shall identify the type of access service for which the accessor has privilege;

b) the **objectDef** component shall specify the object classes for which privilege is assigned to the holder of the attribute certificate for the particular service type. It shall have one element for each object class for which privilege has been assigned. For this access service, the privilege holder has no access to object classes not listed by the **objectDef** component.

7.3.2 Selection of objects

The selection of objects for which the accessor has privileges are specified by an instance of the **ObjectSel** data type.

```
ObjectSel ::= SEQUENCE {
  objecClass         OBJECT-CLASS.&id,
  objSelect          CHOICE {
    allObj        [0]  TargetSelect,
    objectNames   [1]  SEQUENCE SIZE (1..MAX) OF SEQUENCE {
      object             CHOICE {
        names         [1]  SEQUENCE SIZE (1..MAX) OF DistinguishedName,
        subtree       [2]  DistinguishedName,
        ... },
      select           TargetSelect,
      ... },
    ... },
  ... }
```

A value of the **ObjectSel** data type shall specify the privilege for each object class for which privilege has been assigned for the accessed service type. It has the following components:

– the **objectClass** component shall specify the object class for which privileges are assigned;

– the **objSelect** component shall specify what objects of the object class for which privilege has been assigned. It has two alternatives:

  a) the **allObj** alternative shall be taken if the assigned privileges apply equally to all objects of the class. It shall hold an instance of the **TargetSelect** data type;

b) the **objectNames** alternative shall be taken if privilege only applies to selected objects of the identified object class. It may consist of multiple elements, where each element has the following components:

i) the **object** component shall specify one or more objects for which privileges have been assigned. It has the following alternatives:

- the **names** alternative shall specify the name of one or more objects for which the privileges apply;
- the **subtree** alternative shall be taken for a group of objects where each object has a distinguished name equal to the distinguished name held by this alternative or has initial name components equal to that distinguished name;

ii) the select component shall hold an instance of the **TargetSelect** data type.

The **TargetSelect** data type has the following syntax:

```
TargetSelect ::= SEQUENCE {
  objOper   ObjectOperations OPTIONAL,
  attrSel   AttributeSel     OPTIONAL,
  ... }
  (WITH COMPONENTS {..., objOper  PRESENT } |
   WITH COMPONENTS {..., attrSel  PRESENT } )
```

The **TargetSelect** data type has the following two optional components, where at least one of them shall be present:

a) the **objOper** component, when present, shall specify the allowed operations on the objects of the object class or selected objects. If it is not present, no operations are allowed on the objects as a whole;

b) the **attrSel** component, when present, shall hold a value of the **AttributeSel** data type (see clause 7.3.3). If this component is not present, no operations on the attributes of the objects are allowed.

7.3.3 Selection of attribute types

```
AttributeSel ::= SEQUENCE {
  attSelect           CHOICE {
    allAttr        [0]  SEQUENCE {
      attrOper1      [0]  AttributeOperations OPTIONAL,
      ... },
    attributes     [1]  SEQUENCE SIZE (1..MAX) OF SEQUENCE {
      select              SEQUENCE SIZE (1..MAX) OF ATTRIBUTE.&id,
      attrOper2      [0]  AttributeOperations OPTIONAL,
      ... },
    ... },
  ... }
```

The **AttributeSel** data type specify attribute types for which the privilege applies. It has the following component:

– the **attSelect** component has two alternatives:

a) the **allAttr** alternative shall be selected if the privilege applies for all attributes of the object(s). It has the following component:

i) the **attrOper1** component shall specify the operations that may performed on attributes;

b) the **attributes** alternative shall be taken, if the privilege applies only to some attributes of the object(s). The accessor has no privilege for the attribute types not listed and it shall not be made aware of such unlisted attribute types. This alternative has the following components:

i) the **select** component shall specify one or more attribute types for which the privileges apply;

ii) the **attrOper2** component shall specify the operations that may performed on attributes.

7.4 Operations on objects as a whole

The following data type is used for specifying permitted operations against an object:

```
ObjectOperations ::= BIT STRING {
  read            (0),
  add             (1),
  modify          (2),
  delete          (3),
  rename          (4),
  discloseOnError (5) }
```

The **read** permission shall be set for the accessor allowed to read information from an object.

The **add** permission shall be set for the accessor allowed to add new objects of the specific object class. It requires **add** permission for all objects of a specific class. For each attribute to be added to the object, the **add** permission shall be granted for the attribute type (see clause 7.5).

The **modify** permission shall be set for the accessor allowed to modify an existing object. The accessor shall have **modify** permission for the object class as a whole or for the named object(s) to be modified. If the accessor adds attributes, it shall have **add** permissions for the attribute types in question. If the accessor deletes attributes, it shall have the **delete** permission for the attribute types in question. If the accessor modifies attributes, it shall have the **modify** permission for the attribute types in question. If the accessor deletes attributes, it shall have the **deleteValue** permission for the attribute types in question. If the accessor replaces attributes, it shall have the **replaceAttribute** permission for the attribute types in question.

The **delete** permission shall be set for the accessor allowed to delete an existing object. The accessor shall have **delete** permission to the object class as a whole or to the named object(s) to be deleted.

The **rename** permission shall be set for the accessor allowed to rename existing object(s). The accessor shall have **rename** permission to the object class as a whole or to the named object(s) to be renamed.

The **discloseOnError** shall be set for the accessor permitted to know the existence of the object when an operation fails.

7.5 Operations on attributes

```
AttributeOperations ::= BIT STRING {
  read             (0),
  compare          (1),
  add              (2),
  modify           (3),
  delete           (4),
  deleteValue      (5),
  replaceAttribute (6),
  discloseOnError  (7) }
```

The **read** permission shall be set for each of the wanted attribute types for the accessor allowed to read such attributes. The accessor shall have **read** permission to the relevant object class as a whole or to the relevant named objects. In addition, it shall have **read** permission to the all attributes of these object classes or it shall have access to the relevant attribute type(s).

The **compare** permission shall be set for the accessor allowed to compare one or more attributes. The privilege shall **read** permission to the object class as a whole or to the named object(s). In addition, it

shall have compare permission access to the all attributes of the allowed object classes or it shall compare permission to the relevant attribute type(s).

The **add** permission shall be set for the accessor allowed to add one or more attributes. The accessor shall have **modify** permission for the object class as a whole or for the named object(s). In addition, it shall have **add** permission to relevant attributes types.

The **modify** permission shall be set for the accessor allowed to modify an attribute of a specific type. In addition, the accessor shall have **modify** permission for the object class as a whole or to the named object(s).

The **delete** permission shall be set for the accessor allowed to delete one or more attributes. In addition, the accessor shall have **modify** permission for the object class as a whole or to the named object(s).

The **deleteValue** permission shall be set for the accessor allowed to delete one or more attribute values from an attribute of the attribute type(s). In addition, the accessor shall have **modify** permission for the object class as a whole or to the named object(s).

The **replaceAttribute** permission shall be set for the accessor allowed to replace an attribute of a given type with an attribute of the same type. In addition, the accessor shall have **modify** permission for the object class as a whole or to the named object(s).

The **discloseOnError** shall be set for the accessor permitted to know the existence of an attribute when an operation fails. In addition, it shall **discloseOnError** permission for the object as a whole.

7.6 Error handling

Errors may be generated as a result of the use of CMS as discussed in Annex A.5. When an error is detected, no further checking is necessary. A CMS error is returned in the result of the access requested.

Errors may also be generated as the result of the actual access request.

The error is reported by an instance of the **AccessdErr** data type defined in clause 8.10.

8 Privilege assertion protocol

8.1 Overview

The following information object set includes all the defined content types represented by information objects defined by this Recommendation.

```
ActContentTypes CONTENT-TYPE ::= {
    privAssignRequest |
    privAssignResult |
    readRequest |
    readResult |
    compareRequest |
    compareResult |
    addRequest |
    addResult |
    deleteRequest |
    deleteResult |
    modifyRequest |
    modifyResult |
    renameRequest |
    renameResult,
    ... }
```

The content types specified by the **ActContentTypes** set, constitute the privilege assertion protocol, which includes a number of different access operations as specified in clauses 8.4 to 8.9.

8.2 Common request components

The following components are included in all requests:

```
CommonReqComp ::= SEQUENCE {
  attrCerts [31]  AttributeCertificates OPTIONAL,
  serviceId [30]  OBJECT IDENTIFIER,
  invokId   [29]  INTEGER,
  ... }

AtributeCertificates ::= SEQUENCE SIZE (1..MAX) OF AttributeCertificate
```

The common request parameters are:

a) the **attrCert** component, when present, shall specify the attribute certificate or the attribute certification path that holds the privilege for the accessor. If this component is absent, the privilege shall be supplied in the end-entity public-key certificate for the accessor;

b) the **serviceId** component shall specify the type of service to be invoked;

c) the **invokId** component shall take the value zero for the first operation invoked and then be incremented by one for each subsequent invoked operation. It should have a range that ensures that the same value is not reused, for a substantial period of time, for the communication between two entities. It shall be provided to detect missing requests and to detect reply attacks. The recipient of a request shall use the same value in the reply to allow the accessor to pair a result with the corresponding request.

8.3 Accessing a service

All the operation types as specified in clauses 8.4 to 8.9 require access to a particular service. The privilege verifier (recipient) shall check that the privilege assigned to the accessor within the associated attribute certificate or public-key certificate permits access to the requested service.

If the accessor does not have the permission to invoke the requested service type or the service provider does not support the requested service type, a **noSuchService** error code shall be returned.

If the accessor has the permission to invoke the service, it shall be checked whether the requested operation is consistent with the type of service and if not, an **invalidOperationForService** error code shall be returned.

8.4 Read operation

A read operation comprises a read request and a corresponding read result.

A read request is carried as an instance of the **readRequest** content type and the read result is carried as an instance of the **readResult** content type.

```
readRequest CONTENT-TYPE ::= {
              ReadRequest
IDENTIFIED BY id-readRequest
```

The accessor uses the **readRequest** content type to read information about a particular object.

```
ReadRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object    [1] DistinguishedName,
  selection [2] InformationSelection,
  ... }
```

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

a) the **object** component shall hold the distinguished name of the object about which information is requested;

b) the **selection** component shall specify the type of information the accessor requests (see clause 8.11).

The read request shall fail if the request specified an unknown object and a **noSuchObject** error code shall be returned

The read request shall fail if the accessor does not have **read** permission for the object according to the privilege assigned to the accessor. If the **read** permission is not granted, an **insufficientAccessRight** error code shall be returned if the accessor has the **discloseOnError** permission for the object, otherwise a **noSuchObject** error code shall be returned.

The **read** permission for the attribute type is required for each attribute to be returned. If the accessor has no **read** permission to a particular attribute type, an attribute of that type is not returned in the result. If the result is that no attributes are returned, the request fails. If the accessor has **discloseOnError** permission for all of the attribute requests, an **insufficientAccessRight** error code shall then be returned, otherwise a **noInformation** error code shall be returned.

```
readResult CONTENT-TYPE ::= {
              ReadResult
IDENTIFIED BY id-readResult }
```

The privilege verifier shall use an instance of the **readResult** content type to return either the requested information or to report an error situation.

```
ReadResult ::= SEQUENCE {
  object    DistinguishedName,
  result    CHOICE {
    success   [0] ObjectInformation,
    failure   [1] AccessdErr,
    ... },
  ... }
```

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

a) the **object** component shall hold the name of the object about which information was requested;

b) the result component shall hold the result of the read request. It has two alternatives:

   – the **success** alternative shall be selected if information is to be returned and shall hold an instance of the **ObjectInformation** data type (see clause 8.12). The returned information is the intersection between what the accessor requested and what information it is allowed to retrieve;

   – the **failure** alternative shall be selected if an error is to be reported.

8.5 Compare operation

A compare operation comprises a compare request and a corresponding compare result.

A compare request is carried as an instance of the **compareRequest** content type and the compare result is carried as an instance of the **compareResult** content type.

```
compareRequest CONTENT-TYPE ::= {
              CompareRequest
IDENTIFIED BY id-compareRequest }
```

An instance of a **compareRequest** content type is used to compare a presented purported value of a particular attribute type against an attribute value of the same type belonging to a particular object.

```
CompareRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object    [1] DistinguishedName,
  purported [2] AttributeValueAssertion,
  ... }
```

The **CompareRequest** data type specifies the actual content and has the following components beyond those defined in clause 8.2:

a) the **object** component shall hold the distinguished name of the object for which an attribute value is to be compared;

b) the **purported** component shall hold an attribute type and attribute value combination to be compared with an attribute of the same type held by the object in question.

The compare request shall fail if the request specifies an unknown object and a **noSuchObject** error code shall be returned.

The compare request shall fail if the accessor does not have **read** permission for the object according to the privilege assigned to the accessor. If the **read** permission is not granted to the object, an **insufficientAccessRight** error code shall be returned if the accessor has the **discloseOnError** permission for the object, otherwise a **noSuchObject** error code shall be returned.

The compare request shall fail if the accessor does not have **compare** permission to the attribute type in question. If the accessor has **discloseOnError** permission for the attribute type, an **insufficientAccessRight** error code shall then be returned, otherwise a **noInformation** error code shall be returned.

```
compareResult CONTENT-TYPE ::= {
              CompareResult
IDENTIFIED BY id-compareResult
```

The privilege verifier shall use an instance of the **compareResult** content type to return either the requested information or to report an error situation.

```
CompareResult ::= SEQUENCE {
  object    DistinguishedName,
  result    CHOICE {
    success   [0] CompareOK,
    failure   [1] AccessdErr,
    ... },
  ... }

CompareOK ::= SEQUENCE {
  matched        [0] BOOLEAN,
  matchedSubtype [1] BOOLEAN OPTIONAL,
  ... }
```

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

a) the **object** component shall hold the distinguished name of the object on which a compare request was to be done;

b) the result component shall hold the result of the access request. It has two alternatives:

   – if the **success** alternative is selected an instance of the **CompareOK** data type shall be returned with the following components:

     i) the **matched** component shall have the value **TRUE** if an attribute of the attribute type or one of its subtypes had a value equal to the one in the request. In addition, the

matchedSubtype component shall be present and have the value **TRUE** for a subtype match. The **matched** component shall have the value **FALSE** if there was no match for either the attribute type or one of its subtypes;

– the **failure** alternative shall be selected if an error is to be returned.

8.6 Add operation

An add operation comprises an add request and a corresponding add result.

An add request is carried as an instance of the **addRequest** content type and the add result is carried as an instance of the **addResult** content type.

```
addRequest CONTENT-TYPE ::= {
              AddRequest
IDENTIFIED BY id-addRequest }
```

An instance of an **addRequest** content type is used to add a new object to the information system.

```
AddRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object    [1] DistinguishedName,
  attr      [2] SEQUENCE SIZE (1..MAX) OF Attribute {{SupportedAttributes}}
                OPTIONAL,
  ... }
```

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

a) the **object** component shall hold the distinguished name of the new object to be added;

b) the **attr** component, when present, shall hold one or more attributes to be associated with the new object.

If the accessor does not have the **add** permission for the object class, an **insufficientAccessRight** error code shall be returned.

If an object already exists with the supplied distinguished name and the accessor has the **discloseOnError** permission, an **objectAlreadyExists** error code shall be returned, otherwise an **insufficientAccessRight** error code shall be returned.

If the accessor does not have the **add** permission for all of the attributes to be included with the object, the request fails. If the accessor has the **discloseOnError** permission for all the attribute types listed, an **insufficientAccessRight** error code shall be returned, otherwise a **noInformation** error code shall be returned.

```
addResult CONTENT-TYPE ::= {
              AddResult
IDENTIFIED BY id-addResult }
```

The privilege verifier shall use an instance of the **addResult** content type to return either the requested information or to report an error situation.

```
AddResult ::= CHOICE {
  success   [0] NULL,
  failure   [1] AccessdErr,
  ... }
```

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

a) the **success** alternative shall be taken, if the object was added;

b) the **failure** alternative shall be taken, if an error is to be returned.

8.7 Delete operation

A delete operation comprises a delete request and a corresponding delete result.

A delete request is carried as an instance of the **deleteRequest** content type and the delete result is carried as an instance of the **deleteResult** content type.

```
deleteRequest CONTENT-TYPE ::= {
              DeleteRequest
IDENTIFIED BY id-deleteRequest }
```

An instance of a **deleteRequest** content type is used to delete an existing object from the information system.

```
DeleteRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object        DistinguishedName,
  ... }
```

The **DeleteRequest** data type specifies the syntax of the actual content and has the following component:

a) the **object** component shall hold the distinguished name of the entry to be deleted.

If the object to be deleted does not exists, then a **noSuchObject** error code shall be returned.

If the accessor does not have the **delete** permission for the object class, an **insufficientAccessRight** error code shall be returned if the accessor has the **discloseOnError** permission for the object, otherwise a **noSuchObject** error code shall be returned.

```
deleteResult CONTENT-TYPE ::= {
              DeleteResult
IDENTIFIED BY id-deleteResult }
```

The privilege verifier shall use an instance of the **deleteResult** content type either to return the requested information or to report an error situation.

```
DeleteResult ::= CHOICE {
  success   [0] NULL,
  failure   [1] AccessdErr,
  ... }
```

An instance of the **DeleteResult** data type has two alternatives:

a) the **success** alternative shall be taken if the object deletion was done;

b) the **failure** alternative shall be taken if an error is to be reported.

8.8 Modify operation

A modify operation comprises a modify request and a corresponding modify result.

A modify request is carried as an instance of the **modifyRequest** content type and the modify result is carried as an instance of the **modifyResult** content type.

```
modifyRequest CONTENT-TYPE ::= {
              ModifyRequest
IDENTIFIED BY id-modifyRequest }
```

An instance of a **modifyRequest** content type is used to modify an existing object.

```
ModifyRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object        DistinguishedName,
  changes       SEQUENCE SIZE (1..MAX) OF ObjectModification,
  select        InformationSelection,
```

```
  ... }

ObjectModification ::= CHOICE {
  addAttribute     [0]  Attribute{{SupportedAttributes}},
  deleteAttribute  [1]  AttributeType,
  addValues        [2]  Attribute{{SupportedAttributes}},
  deleteValues     [3]  Attribute{{SupportedAttributes}},
  replaceAttribute [4]  Attribute{{SupportedAttributes}},
  ... }
```

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

a) the **object** component shall hold the distinguished name of the object to be modified:
   - if the object does not exists, a **noSuchObject** error code shall be returned;
   - if the accessor does not have **modify** permission to the object, an **insufficientAccessRight** error code shall be returned if the accessor have the **discloseOnError** permission for the object, otherwise a **noSuchObject** error code shall be returned.

b) the **changes** component shall hold information for modifying one or more attributes:
   - the **addAttribute** alternative shall hold a new attribute to be added:
     i) if the accessor does not have the **add** permission for the attribute type, an **insufficientAccessRight** error code shall be returned;
     ii) if an attribute of the type already exists, an **attributeAlreadyExists** error code shall be returned if the accessor has **discloseOnError** permission for the attribute type, otherwise an **insufficientAccessRight** error code shall be returned;
   - the **deleteAttribute** alternative shall identify the attribute to be deleted:
     i) if the accessor does not have the **delete** permission for the attribute type, an **insufficientAccessRight** error code shall be returned;
     ii) if an attribute of the type does not exist, a **noSuchAttribute** error code shall be returned;
   - the **addValues** alternative shall identify an existing attribute by the attribute type. The values to be added to the attribute are the values included in this alternative.
     i) if no attribute of the given type is held by the object, a **noSuchAttribute** error code shall be returned if the accessor has the **discloseOnError** permission, otherwise an **insufficientAccessRight** error code shall be returned;
     ii) if the accessor does not have the **addValue** permission, an **insufficientAccessRight** error code shall be returned;
     iii) if an attempt is made to add an already existing value, an **attributeValueAlreadyExists** error code shall be returned if the accessor has the **discloseOnError** permission, otherwise an **insufficientAccessRight** error code shall be returned;
   - this **deleteValues** alternative shall identify an attribute by the attribute type. The values to be deleted from the attribute are the values included in this alternative.
     i) if no attribute of the given type is held by the object, a **noSuchAttribute** error code shall be returned if the accessor has the **discloseOnError** permission, otherwise an **insufficientAccessRight** error code shall be returned;
     ii) if the accessor does not have the **deleteValue** permission, an **insufficientAccessRight** error code shall be returned if the accessor has the

discloseOnError permission, otherwise a **noSuchAttributeValue** error code shall be returned;

iii) if the accessor tries to delete an attribute value that does not exist, a **noSuchAttributeValue** error code shall be returned;

– the **replaceAttribute** alternative shall replace an existing attribute with a new attribute of the same type.

i) if no attribute of the given type is held by the object, a **noSuchAttribute** error code shall be returned if the accessor has the **discloseOnError** permission, otherwise an **insufficientAccessRight** error code shall be returned;

ii) if the accessor does not have the **replaceAttribute** permission, an **insufficientAccessRight** error code shall be returned if the accessor has the **discloseOnError** permission, otherwise a **noSuchAttribute** error code shall be returned.

```
modifyResult CONTENT-TYPE ::= {
              ModifyResult
IDENTIFIED BY id-modifyResult }
```

The privilege verifier shall use an instance of the **modifyResult** content type either to return the requested information or to report an error situation.

```
ModifyResult ::= SEQUENCE {
  result    CHOICE {
    success   [0] ObjectInformation,
    failure   [1] AccessdErr,
    ... },
  ... }
```

An instance of the **ModifyResult** data type has two alternatives:

a) the **success** alternative shall be taken if the object modification was done;

b) the **failure** alternative shall be taken if an error is to be returned.

8.9 Rename object operation

A rename object operation comprises a rename request and a corresponding rename result.

A rename request is carried as an instance of the **renameRequest** content type and the rename result is carried as an instance of the **renameResult** content type.

```
renameRequest CONTENT-TYPE ::= {
              RenameRequest
IDENTIFIED BY id-renameRequest }
```

An instance of a **renameRequest** content type is used to change the name of an existing object.

```
RenameRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object        DistinguishedName,
  new           DistinguishedName,
  ... }
```

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

a) the **object** component shall specify the current distinguished name of the object to be renamed;

b) the **new** component shall provide the new distinguished name for the object.

If the object to be renamed does not exists, then a **noSuchObject** error code shall be returned.

If the accessor does not have the **rename** permission for the named object, an **insufficientAccessRight** error code shall be returned if the accessor has the **discloseOnError** permission for the named object, otherwise a **noSuchObject** error code shall be returned.

```
renameResult CONTENT-TYPE ::= {
              RenameResult
IDENTIFIED BY id-renameResult }

RenameResult ::= SEQUENCE {
  result    CHOICE {
    success   [0] NULL,
    failure   [1] AccessdErr,
    ... },
  ... }
```

An instance of the **RenameResult** data type has two alternatives:

a) the **success** alternative shall be taken if the object modification was done;

b) the **failure** alternative shall be taken if an error is to be returned.

8.10 Error handling

When an exception condition occurs during handling of a request, an error code has to be returned by the recipient by including an instance of the **AccessErr** data type in the result.

```
AccessdErr ::= CHOICE {
  cmsErr     [0] CmsErr,
  ActErr   [1] PbactErr,
  ... }
```

The **cmsErr** alternative shall be taken if an exception condition occurred while evaluating the CMS defined content types (see clause A.5).

The **pbactErr** alternative shall be taken if there was no error in detected when evaluating instances of the CMS content types, but an error was detected in an encapsulated instance of a PBACT specific content type.

8.11 Information selection

The **InformationSelection** data type is used for specifying what information is requested by a read or modify request.

```
InformationSelection ::= SEQUENCE {
  attributes      CHOICE {
    allAttributes   [0]  NULL,
    select          [1]  SEQUENCE SIZE (1..MAX) OF ATTRIBUTE.&id,
    ... },
  infoTypes        ENUMERATED {
    attributeTypesOnly     (0),
    attributeTypeAndValues (1),
    ... },
  ...}
```

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

a) the **attributes** component shall specify what attributes should be returned. It has two alternatives:

   – the **allAttributes** alternative shall be taken if the accessor wants all information about the object; or

   – the **select** alternative shall be taken when only a selected set of attributes are requested;

b) the **infoTypes** component has the following enumeration items:

- the **attributeTypesOnly** enumeration item shall be taken if only attribute types are to be returned. In this case, the accessor shall have *read* permission for the attribute type according to the current privilege. If that is not the case, the attribute type is removed from the result. If that results in no information being returned, the request fails;
- the **attributeTypesAndValues** enumeration item shall be taken if both type and values shall be returned for the privilege inforce. In this case, the accessor shall have *read* permission for the attribute type according to the current privilege. If that is not the case, the attribute type and value are removed from the result. If that results in no information being returned, the request fails.

8.12 Object information

When information about an object is to be returned, it shall be returned as an instance of the following data type:

```
ObjectInformation ::= SEQUENCE {
  object  DistinguishedName,
  info    CHOICE {
    attr    SET SIZE (1..MAX) OF Attribute {{SupportedAttributes}},
    type    SET SIZE (1..MAX) OF AttributeType },
  ... }
```

The **object** component shall hold the distinguished name of the object for which information is returned.

The **info** component shall hold a set of attributes holding the requested information or set of attribute types.

If there is no information to returned, the request shall fail.

8.13 Defined error codes

The error codes for the specific PBACT content types are defined here.

```
PbactErr ::= ENUMERATED {
  noSuchService,
  invalidOperationForService,
  insufficientAccessRight,
  noSuchObject,
  noSuchAttribute,
  noSuchAttributeValue,
  objectAlreadyExists,
  attributeAlreadyExists,
  attributeValueAlreadyExists,
  noInformation,
  ... }
```

a) the **noSuchService** error code shall be returned if the accessor specifies a service for which it has no permission and is not allowed to know about or which is not supported;

b) the **invalidOperationForService** error code shall be returned if a requested operation is not relevant for the service requested;

c) the **insufficientAccessRight** error code shall be returned when the accessor requests service to which it has no permission or when it wants to perform an operation for which it has no permission according the service to be accessed;

d) the **noSuchObject** error code shall be returned if an accessor tries to access a non-existing object or an object it is not allowed to know the existence of;

e) the **noSuchAttribute** error code shall be returned if an accessor tries to access a non-existing attribute or an attribute it is not allowed to know the existence of;

f) the **noSuchAttributeValue** error code shall be returned if an accessor tries to access a non-existing attribute value or an attribute value it is not allowed to know the existence of;

g) the **objectAlreadyExists** error code shall be returned if an attempt is made to add an object with a distinguished name equal to a distinguished name of an already existing object providing that the accessor has permission to know the existence of that object;

h) the **attributeAlreadyExists** error code shall be returned if an attempt is made to add an attribute of a type that already exists within the object in question providing that the accessor has permission to know the existence of that attribute type within the object.

i) the **attributeValueAlreadyExists** error code shall be returned if an attempt is made to add an attribute to an object that already holds an attribute of the same type providing the accessor has permission to know the existence of that attribute;

j) the **noInformation** error code shall be returned if information is requested but either not available or the accessor has no permission to know the existence of that data.

9 Privilege assignment protocol

9.1 Scope of protocol

The privilege assignment protocol is used for assigning privileges:

a) from an SOA to an intermediate AA for further delegation;

b) from an SOA directly to an entity using the assigned privileges to assert those privileges;

c) from an AA directly to an entity using the assigned privileges to assert those privileges; and

d) when multiple AAs are in the path between SOA and the privilege holder, from one AA to another AA.

NOTE – It is recommended to make the delegation path as short as possible.

9.2 Content types

The privilege assignment protocol makes use of two content type: one content type for assigning privileges and one content type for confirming the assignment.

9.2.1 The privilege assignment request content type

A privilege assignment request is carried in an instance of the **privAssignRequest** content type.

```
privAssignRequest CONTENT-TYPE ::= {
                PrivAssignRequest
IDENTIFIED BY id-privAssignRequest }
```

The syntax of the actual content is specified by the following data type:

```
PrivAssignRequest ::= SEQUENCE {
  attrCerts   AttributeCertificates OPTIONAL,
  ... }
```

This data type only has one component holding a sequence of attribute certificates. If the privilege assignment request is sent from an SOA, the sequence consists of only one attribute certificate. If there is a single AA in between the SOA and the privilege holder, the request from the AA to the privilege holder shall include both the attribute certificate issued by the SOA and the attribute certificate issued by the AA. One more attribute certificate is required for each additional AA between the SOA and the privilege holder.

9.2.2 The privilege assignment result content type

A privilege assignment result is carried in an instance of the **privAssignResult** content type.

```
privAssignResult CONTENT-TYPE ::= {
              PrivAssignResult
IDENTIFIED BY id-privAssignResult }
```

The syntax of the actual content is specified by the following data type:

```
PrivAssignResult ::= SEQUENCE {
  result  CHOICE {
    success  NULL,
    failure  PrivAssignErr },
  ... }

PrivAssignErr ::= CHOICE {
--cmsErr     [0] CmsErr,
  assignErr  [1] AssignErr,
  ... }
```

9.2.3 Defined error codes

```
AssignErr ::= ENUMERATED {
  invalidAttributeCertificate (0),
  invalidDelegationPath
  invalidPublicKeyCertificate
    ... }
```

Annex A

Object identifier allocation for the ITU-T 1080-series

(This annex forms an integral part of this Recommendation.)

A.1 Top level of object identifier tree

~~Annex A of [ITU-T X.1081] allocates arcs below t~~The following arc~~h~~ is allocated for telebiometrics~~, which is~~:

```
id-telebio OBJECT IDENFIER ::= { joint-iso-itu-t(2) telebiometrics(42) }
```

Below this arc, ~~[ITU-T 1081] allocates~~ the following arc is allocated for telehealth protocol related objects:

```
id-thprot~~telehealth~~ OBJECT IDENTIFIER ::= { id-telebio thprot~~telehealth~~(~~3~~10) }
```

~~[ITU-T X.1080.1] has allocated several arcs below the id-th arc. Arc '0' is allocated for ASN.1 modules defined within [ITU-T 1080.1]. Other arcs are allocated for entity categories. This Recommendation specifies that arc '0' shall also be used for allocation of object identifiers for the ITU-T X.1080 series in general. To avoid collision, the value 10 is used for allocation of object identifiers for the ITU-T X.1080 series.~~

```
~~id-telehelth OBJECT IDENTIFIER ::= { id-th all(0) telehealth(10) }~~
```

The following arcs are allocated for the different parts of the ITU-T X.1080-series:

```
id-x1080-0 OBJECT IDENTIFIER ::= { id-~~telehel~~thprot part0(0) }
id-x1080-1 OBJECT IDENTIFIER ::= { id-~~telehel~~thprot part1(1) }
id-x1080-2 OBJECT IDENTIFIER ::= { id-~~telehel~~thprot part2(2) }
----
```

An arc for a particular part of the ITU-T X.1080-series is divided as follows:

– modules have the arc '0';

– CMS content types have the arc '1';

– attribute types have the arc~~h~~ '2'.

A particular part may allocate additional arcs according to its needs.

For this Recommendation, the following arc is allocated for modules:

```
id-x1080-0-module OBJECT IDENTIFIER ::= { id-x1080-0 module(0) }
```

The following arc is allocated for CMS content types:

```
id-x1080-0-Cont   OBJECT IDENTIFIER ::= { id-x1080-0 cmsCont(1) }
```

The following arc is allocated for attribute types used for assigning privileges:

```
id- x1080-0-attr   OBJECT IDENTIFIER ::= { id-x1080-0 prAttr(2) }
```

A.2 Object identifiers for CMS content types

The following object identifiers are allocated for the content types defined for the privilege assignment protocol and for the privilege assessment protocol:

```
id-privAssignReq  OBJECT IDENTIFIER ::= { id-x1080-0-Cont privAssignRequest(1) }
id-privAssignRes  OBJECT IDENTIFIER ::= { id-x1080-0-Cont privAssignResult(2) }
id-readRequest    OBJECT IDENTIFIER ::= { id-x1080-0-Cont readRequest(3) }
id-readResult     OBJECT IDENTIFIER ::= { id-x1080-0-Cont readResult(4) }
id-compareRequest OBJECT IDENTIFIER ::= { id-x1080-0-Cont compareRequest(5) }
```

```
id-compareResult  OBJECT IDENTIFIER ::= { id-x1080-0-Cont compareResult(6) }
id-addRequest     OBJECT IDENTIFIER ::= { id-x1080-0-Cont addRequest(7) }
id-addResult      OBJECT IDENTIFIER ::= { id-x1080-0-Cont addResult(8) }
id-deleteRequest  OBJECT IDENTIFIER ::= { id-x1080-0-Cont deleteRequest(9) }
id-deleteResult   OBJECT IDENTIFIER ::= { id-x1080-0-Cont deleteResult(10) }
id-modifyRequest  OBJECT IDENTIFIER ::= { id-x1080-0-Cont modifyRequest(11) }
id-modifyResult   OBJECT IDENTIFIER ::= { id-x1080-0-Cont modifyResult(12) }
id-renameRequest  OBJECT IDENTIFIER ::= { id-x1080-0-Cont renameRequest(13) }
id-renameResult   OBJECT IDENTIFIER ::= { id-x1080-0-Cont renameResult(14) }
```

A.3 Object identifiers for privilege attribute types

```
id-at-accessSer   OBJECT IDENTIFIER ::= { id-pbactPrivAttrid-x1080-0-attr 1 }
```

Annex B

Cryptographic message syntax profile

(This annex forms an integral part of this Recommendation.)

B.1 General

Cryptographic message syntax (CMS) is defined in [IETF RFC 5652]. It defines communication capabilities that allow for data integrity, authentication and confidentiality. [IETF RFC 5083] add additional specifications. [IETF RFC 5911] and [IETF RFC 6268] provides new ASN.1 modules for CMS. This annex provides a profile of CMS for use by telebiometrics specifications by references to these specifications.

CMS is a versatile specification to be used in many different environments. This profile does not make use of all the CMS capabilities. This profile includes the CMS data types used by this Recommendation and other telebiometrics specifications. These other telebiometrics specifications may refer to this annex for their use of CMS.

It is not the intention to make a specification for an implementation that is not compliant with the IETF RFCs, but only to discuss those aspects of CMS that are relevant to telebiometrics specifications. Appendix I provides an informal ASN.1 module reflecting the telebiometrics use of CMS.

CMS defines different content types to be used for different purposes. The telebiometrics specifications make use of the **signedData** content type for providing authentication and integrity, they make use of the **envelopedData** content type for providing encryption and thereby confidentiality and they make use of the **ct-authEnvelopedData** content type. The **signedData** content type is used when digital signing is required, while the **envelopedData** content type is used when confidentiality is required. When used, an instance of the **envelopedData** content type is encapsulated in an instance of the **signedData** content type. The **ct-authEnvelopedData** content type is used when multiple messages constitute a particular task.

Not all aspects of the above content types are used by the telebiometrics specifications. This annex therefore provides a profile for the use of CMS in telebiometrics. For easy reference, the relevant aspects of CMS are included here.

An instance of a content type defined by telebiometrics specifications is encapsulated in an instance of the **envelopedData** content type, if confidentiality is required; otherwise, it is encapsulated in an instance of the **signedData** content type. Alternatively, such a content type instance may be included in an instance of the **ct-authEnvelopedData** content type.

A content type, according to [IETF RFC 6268], is defined using the following information object class:

```
CONTENT-TYPE ::= TYPE-IDENTIFIER
```

The **CONTENT-TYPE** information object class is equivalent to the ASN.1 built information object class **TYPE-IDENTIFIER**. A **CONTENT-TYPE** information object is used to bind the content type identified by an object identifier to the abstract syntax of the content.

The following **ContentInfo** data type provides the general syntax for a content type:

```
ContentInfo ::= SEQUENCE {
  contentType  CONTENT-TYPE.&id ({TelebSupportedcontentTypes}),
  content      CONTENT-TYPE.&Type
          ({TelebSupportedcontentTypes}{@contentType})OPTIONAL,
  ... }
```

```
TelebSupportedcontentTypes CONTENT-TYPE ::=
  { signedData | envelopedData | ct-authEnvelopedData, ...}
```

The supported content types are the **signedData** content type, the **envelopedData** content type, the **ct-authEnvelopedData** content type and the set of content types defined by a particular telebiometrics specification.

CMS requires that a CMS version be specified for a data type to indicate the specific syntax used for that data type. The following versions are defined:

```
CMSVersion ::= INTEGER{ v0(0), v1(1), v2(2), v3(3), v4(4), v5(5) }
```

[IETF RFC 6268] defines the following parameterized data type used throughout the specifications:

```
Attributes { ATTRIBUTE:AttrList } ::=
             SET SIZE (1..MAX) OF Attribute {{ AttrList }}
```

B.2 Use of the signedData content type

The following content type is specified in clause 5 of [IETF RFC 5652]. Using a slightly modified notation reflecting its use in telebiometrics, it is specified as follows:

```
signedData CONTENT-TYPE ::= {
                SignedData
  IDENTIFIED BY id-signedData }

SignedData ::= SEQUENCE  {
  version               CMSVersion (v3),
  digestAlgorithms      SET (SIZE (1)) OF AlgorithmIdentifier
                          {{Teleb-Hash-Algorithms}},
  encapContentInfo      EncapsulatedContentInfo,
  certificates     [0]  IMPLICIT SET (SIZE (1..MAX)) OF Certificate OPTIONAL,
  crls             [1]  IMPLICIT RevocationInfoChoices OPTIONAL,
  signerInfos           SignerInfos,
  ... }
```

NOTE 1 – [IETF RFC 6268] uses a somewhat modified version of the **AlgorithmIdentifier** data type. However, this profile uses the **AlgorithmIdentifier** data type as defined in [ITU-T X.509].

The **version** component shall take the value **v3** according to clause 5.1 of [IETF RFC 5652].

The **digestAlgorithms** component shall consist of a single element specifying a hashing algorithm from the set of applicable hashing algorithms as defined by the applicable telebiometrics specification.

The following definition allows any hash algorithm to be included.

```
Teleb-Hash-Algorithms ALGORITHM ::= {...}
```

NOTE 2 – This profile does not mandate a specific hash algorithm. Reference specifications or implementer's' agreements may replace the dots with a set of hash algorithms to be supported in a specific environment.

NOTE 3 – CMS allows for multiple digital signatures and thereby for multiple hash algorithms. Multiple digital signatures are not relevant for telebiometrics specifications.

The **encapContentInfo** component shall hold an instance of the following data type:

```
EncapsulatedContentInfo ::= SEQUENCE {
  eContentType     CONTENT-TYPE.&id({envelopedData, ...}),
  eContent     [0] EXPLICIT OCTET STRING
    (CONTAINING CONTENT-TYPE.&Type({envelopedData, ...}
     {@eContentType})) OPTIONAL }
```

This data type has the following components:

a) the **eContentType** component shall hold the object identifier identifying the encapsulated content type. This component shall hold the identity of the **envelopedData** content type, if encryption is required. If encryption is not required, it shall hold one of the content types specified by the relevant telebiometrics specification;

b) the **econtent** component shall hold the actual encapsulated content wrapped in an octet-string. It shall always be present.

NOTE 4 – This component is defined as optional. However, all relevant content types have a defined content.

The **certificates** component shall hold the public-key certificates sufficient to establish a single certification path as specified by the **PkiPath** data type defined in [ITU-T X.509].

The **crls** component is not relevant for the telebiometrics specifications and shall be absent.

The **signerInfos** component shall hold an instance of the **SignerInfos** data type:

```
SignerInfos ::= SET (SIZE (1)) OF SignerInfo

SignerInfo ::= SEQUENCE {
  version                 CMSVersion (v1),
  sid                     SignerIdentifier,
  digestAlgorithm         AlgorithmIdentifier {{Teleb-Hash-Algorithms}},
  signedAttrs        [0]  IMPLICIT Attributes{{SignedAttributes}} OPTIONAL,
  signatureAlgorithm      AlgorithmIdentifier {{Teleb-Signature-Algorithms}},
  signature               SignatureValue,
  unsignedAttrs      [1]  IMPLICIT Attributes {{UnsignedAttributes}} OPTIONAL,
  ... }

SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber    IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier,
  ...}

IssuerAndSerialNumber ::= SEQUENCE {
  issuer       Name,
  serialNumber CertificateSerialNumber }

SignedAttributes ATTRIBUTE ::= { contentType | messageDigest, ... }

Teleb-Signature-Algorithms ALGORITHM ::= {...}

SignatureValue ::= OCTET STRING

UnsignedAttributes ATTRIBUTE ::= {...}
```

This profile only supports one signer, so an instance of the **SignerInfos** data type shall have one and only one element. The **SignerInfo** data type has the following components:

a) the **version** component shall specify **v1** according to [IETF RFC 5652];

b) the **sid** component shall identify the end-entity public-key certificate of the signer and shall hold an instance of the **SignerIdentifier** data type. This data type specifies two alternatives

   – the **issuerAndSerialNumber** alternative identifies the end-entity public-key certificate by specifying the distinguished name of the issuing CA and the public-key certificate serial number. This alternative shall always be taken;

   – the **subjectKeyIdentifier** alternative shall not be taken;

c) the **digestAlgorithm** component shall have the same value as the one used in **digestAlgorithms** component of the **SignerInfo** data type;

d) the **signedAttrs** component shall hold a list of signed attributes. [IETF RFC 5652] requires that at least instances of the **contentType** and **messageDigest** attribute types shall be included. This profile does not any require additional attribute to be included, but referencing specifications may add to the list;

e) the **signatureAlgorithm** component shall hold the signature algorithm used for creating the digital signature held by the **signature** component;

NOTE 5 – This profile does not mandate a specific signature algorithm. Reference specifications or implementer's' agreements may replace the dots with a set of signature algorithms to be supported for a specific telebiometrics specification.

f) the **signature** and the **unsignedAttrs** components as specified by [IETF RFC 5652].

B.3 Use of envelopedData content type

B.3.1 General

The **envelopedData** content type allows for encryption of data. This requires establishment of shared symmetric keys. [IETF RFC 5652] provides for different techniques for generating such symmetric keys. The key agreement technique, known as the Diffie-Hellman (DH) key agreement method is required by this profile. The DH key agreement method is specified [IETF RFC 2631] for non-elliptic curve technique. [IETF RFC 5753] provides specifications for use of elliptic curve techniques.

The DH method results in a shared secret that may be used as keying material that allows generation of shared, symmetric keys. This profile recognizes two modes of DH operation: the ephemeral-static mode and the static-static mode.

The ephemeral-static mode requires that the recipient have a public-key certificate with a DH public key as certified by the issuing CA. This public-key certificate shall be available to the sender. The sender creates a new DH key pair for each message it sends. In this way, the shared secret becomes different for each message.

The static-static mode requires that each of the communicating entities have a certified DH public-key certificate. As this mode would result in the same shared secret for each message, some random user keying material has to be supplied by the sender to get different keying material for each message.

This profile requires that the ephemeral-static mode shall be supported.

Both methods require both entities in a communication to have the certified DH end-entity public-key certificate of its partner, as communications go in both directions.

The following content type is specified in clause 6 of [IETF RFC 5652] and updated by [IETF RFC 6268]:

```
envelopedData CONTENT-TYPE ::= {
                EnvelopedData
  IDENTIFIED BY id-envelopedData }

EnvelopedData ::= SEQUENCE {
  version                    CMSVersion(v0 | v2),
  originatorInfo         [0] IMPLICIT OriginatorInfo OPTIONAL,
  recipientInfos             RecipientInfos,
  encryptedContentInfo       EncryptedContentInfo,
  ...,
  [[2: unprotectedAttrs  [1] IMPLICIT Attributes
       {{UnprotectedAttributes}} OPTIONAL ]] }

UnprotectedAttributes ATTRIBUTE ::=
  { aa-CEKReference | aa-CEKMaxDecrypts | aa-KEKDerivationAlg }
```

The **EnvelopedData** component has the following components:

a) the **version** component shall according to [IETF RFC 5652] have the value **v2** if the **unprotectedAttrs** component is present, otherwise it shall have the value **v0**;

b) the **originatorInfo** component shall be absent;

c) the **recipientInfos** component shall hold an instance of the **RecipientInfos** data type as specified in clause B.3.2;

d) the **encryptedContentInfo** component shall hold an instance of the **EncryptedContentInfo** data type as specified in clause B.3.4;

e) the **unprotectedAttrs** component is required if it is expected that the next to be transmitted in the direction in question is an instance of the **ct-authEnvelopedData** content type, otherwise it may be absent.

B.3.2 Recipient information

The **RecipientInfos** data type allows for multiple instances of the **RecipientInfo** data type. However, for the purpose of this profile, it shall be limited to a single instance. An instance of the **RecipientInfo** data type specifies different alternatives on how to establish a shared secret between the two communicating parties.

```
RecipientInfos ::= SET SIZE (1) OF RecipientInfo

RecipientInfo ::= CHOICE {
  ktri      KeyTransRecipientInfo,
  kari  [1] KeyAgreeRecipientInfo,
  kekri [2] KEKRecipientInfo,
  pwri  [3] PasswordRecipientinfo,
  ori   [4] OtherRecipientInfo,
  ... }
```

This profile only makes use of two of the **RecipientInfo** alternatives as defined in [IETF RFC 5652]:

a) the **kari** alternative is the only alternative to be used for the **envelopedData** content type. The **KeyAgreeRecipientInfo** data type provides information required for establishment of a shared secret as detailed in clause B.3.3;

b) the **kekri** alternative is the only alternative to be used for the **ct-authEnvelopedData** content type. The **KEKRecipientInfo** data type provides information required for establishment of a shared secret as detailed in clause B.4.

B.3.3 Key agreement

```
KeyAgreeRecipientInfo ::= SEQUENCE {
  version                CMSVersion (v3),
  originator         [0] EXPLICIT OriginatorIdentifierOrKey,
  ukm                [1] EXPLICIT UserKeyingMaterial OPTIONAL,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  recipientEncryptedKeys RecipientEncryptedKeys,
  ... }

OriginatorIdentifierOrKey ::= CHOICE {
  issuerAndSerialNumber    IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier,
  originatorKey        [1] OriginatorPublicKey,
  ... }

OriginatorPublicKey ::= SEQUENCE {
  algorithm AlgorithmIdentifier {{SupportedDHPublicKeyAlgorithms}},
  publicKey BIT STRING,
  ... }

SupportedDHPublicKeyAlgorithms ALGORITHM ::= {...}
```

```
UserKeyingMaterial ::= OCTET STRING (SIZE (64))

KeyEncryptionAlgorithmIdentifier ::=
    AlgorithmIdentifier{{SupportedKeyIncryptAlgorithms}}

SupportedKeyIncryptAlgorithms ALGORITHM ::= {...}

RecipientEncryptedKeys ::= SEQUENCE (SIZE (1)) OF RecipientEncryptedKey

RecipientEncryptedKey ::= SEQUENCE {
  rid          KeyAgreeRecipientIdentifier,
  encryptedKey EncryptedKey }

KeyAgreeRecipientIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
--rKeyId             [0] IMPLICIT RecipientKeyIdentifier,
  ... }

EncryptedKey ::= OCTET STRING
```

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

a) the **version** component shall according to [IETF RFC 5652] take the value **v3**;

b) the **originator** component shall hold an instance of the **OriginatorIdentifierOrKey** data type with the following alternatives:

- the **issuerAndSerialNumber** alternative shall be taken if the static-static method is used. It shall then hold an instance of the **IssuerAndSerialNumber** data type. This data type shall identify the sender's DH public-key certificate.
  - i) the **issuer** component shall hold the distinguished name of the issuing CA and shall be equal to the **issuer** component of the public-key certificate in question;
  - ii) the **serialNumber** component shall be equal to the **serialNumber** component of the public-key certificate in question;
- the **subjectKeyIdentifier** alternative shall not be taken;
- the **originatorKey** alternative shall be taken for the ephemeral-static DH method. It shall then hold an instance of the **OriginatorPublicKey** data type with the following components:
  - i) the **algorithm** component shall hold a reference to the DH public-key algorithm used;

NOTE 1 – This profile does not mandate a specific DH public-key algorithm. Reference specifications or implementer's' agreements may replace the dots with a set of DH public-key algorithms to be supported in a specific environment.

  - ii) the **publicKey** component shall hold the DH public key as generated by the sender. The sender shall generate a new DH key pair for each use of this content type.

From the local private key and the public key of the recipient, the sender can generate the shared secret. The recipient shall generate an identical shared secret by using its private key together with the sender's public key as provided in the **OriginatorPublicKey** or in the **IssuerAndSerialNumber** data type.

c) the **ukm** component shall be present when the static-static method is used and shall the hold an instance of the **UserKeyingMaterial** data type.

From the shared secret, the value in the **ukm** component (if relevant) and some other information as specified in [IETF RFC 2631] both parties generate the so-called key-encryption key (KEK). This key is subsequently used for encrypting the content encryption key (CEK) generated by the sender. This technique is referred to as key wrapping;

d) the **keyEncryptionAlgorithm** component shall specify the key wrapping algorithm and shall hold an instance of the **KeyEncryptionAlgorithmIdentifier** data type;

NOTE 2 – This profile does not mandate a specific set of key wrapping algorithms. New algorithms may be defined in the future. Advanced encryption standard (AES) Key Wrap Algorithms are defined by [IETF RFC 3394]. Reference specifications or implementer's' agreements may replace the dots with a set of wrap algorithms to be supported in a specific environment.

e) the **recipientEncryptedKeys** component shall hold an instance of the **RecipientEncryptedKeys** data type. Such an instance shall only consist of a single element, i.e., a single instance of the **RecipientEncryptedKey** data type. This data type has the following two components:

- the **rid** component shall hold the identification of the recipient by its end-entity public-key certificate;
- the **encryptedKey** component shall hold the encrypted CEK used for encrypting the content, as discussed under item c).

B.3.4 Reuse of CMS content encryption keys

If the CEK is to be used for a subsequent **ct-authEnvelopedData** content type instance as specified in [IETF RFC 3185], the appropriate reference information shall be entered into the unprotected attributes as discussed in clause B.3.1. This information shall be retained by both parties. If high level of security is required, the attribute of type **aa-CEKMaxDecrypts** should have the value '1' or be omitted.

B.3.5 Encrypted content information

An instance of the **EncryptedContentInfo** data type holds the encrypted encapsulated content.

```
EncryptedContentInfo ::= SEQUENCE {
  contentType                 CONTENT-TYPE.&id ({EncryptedContentSet}),
  contentEncryptionAlgorithm SEQUENCE {
    algorithm                   ALGORITHM.&id ({SymmetricEncryptionAlgorithms}),
    parameter                   ALGORITHM.&Type
              ({SymmetricEncryptionAlgorithms}{@.algorithm})} OPTIONAL,
  encryptedContent       [0] IMPLICIT EncryptedContent OPTIONAL,
  ... }

EncryptedContentSet CONTENT-TYPE ::= {...}

SymmetricEncryptionAlgorithms ALGORITHM ::= {...}

EncryptedContent ::= OCTET STRING
```

The **encryptedContentInfo** component of the **EnvelopedData** data type shall hold an instance of the **EncryptedContentInfo** data type:

a) the **contentType** component shall hold the content type for the encrypted content. The list of possible content types is those for which encryption is an option;

b) the **contentEncryptionAlgorithm** and the **encryptedContent** components as required by [IETF RFC 5652].

B.4 Use of the authenticated-enveloped-data content type

B.4.1 General

As specified in [ITU-T X.1080.1], protocols for telebiometrics general consists of a set-up exchange to establish a session followed by multiple exchanges of information and ending with the termination of the session. In such an environment, it may not be necessary to establish a new key encryption key for each message.

[IETF RFC 5083] specifies a content type **ct-authEnvelopedData** not included in [IETF RFC 5652]. This content type allows for use of efficient authenticated encryption techniques. This profile makes use of the AES-GCM algorithms a defined by [IETF RFC 5084] together with the reuse of the CEK as defined in [IETF RFC 3185]. For details, these specifications should be consulted.

The **ct-authEnvelopedData** content type is defined as:

```
ct-authEnvelopedData CONTENT-TYPE ::= {
                AuthEnvelopedData
  IDENTIFIED BY id-ct-authEnvelopedData }

AuthEnvelopedData ::= SEQUENCE {
  version                   CMSVersion (v0),
  originatorInfo       [0] IMPLICIT OriginatorInfo OPTIONAL,
  recipientInfos            RecipientInfos,
  authEncryptedContentInfo EncryptedContentInfo,
  authAttrs            [1] IMPLICIT Attributes {{AuthAttributes}} OPTIONAL,
  mac                       MessageAuthenticationCode,
  unauthAttrs          [2] IMPLICIT Attributes {{UnauthAttributes}} OPTIONAL }

AuthAttributes ATTRIBUTE ::= {...}

MessageAuthenticationCode ::= OCTET STRING

UnauthAttributes ATTRIBUTE ::=
  { aa-CEKReference | aa-CEKMaxDecrypts | aa-KEKDerivationAlg }
```

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

a) the **version** component shall according to [IETF RFC 5083] have the value **v0**;

b) the **originatorInfo** component shall be absent;

c) the **recipientInfos** component shall hold an instance of the **RecipientInfos** data type. This data type is described in clause B.3.2. The **kekri** alternative, in addition to **kari** alternative, is relevant for this content type. When the **kekri** alternative is taken, this alternative shall hold an instance of the **KEKRecipientInfo** data type as specified in clause B.4.2;

d) the **authEncryptedContentInfo** component shall hold an instance of the **EncryptedContentInfo** data type as specified in clause B.3.5;

e) the **authAttrs** component, when present, shall hold a set of attributes to be under authentication protection;

f) the **mac** component shall hold the generated message authentication code (MAC);

g) the **unauthAttrs** component hold attributes of the same type as specified in clause B.3.1, item e). If it is known that the content type instance is the last one for the session in question for the direction, then this component may be absent.

B.4.2 KEK recipient information

```
KEKRecipientInfo ::= SEQUENCE {
  version                CMSVersion (v4),
  kekid                  KEKIdentifier,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey           EncryptedKey }

KEKIdentifier ::= SEQUENCE {
  keyIdentifier OCTET STRING,
  date          GeneralizedTime OPTIONAL,
  other         OtherKeyAttribute OPTIONAL,
  ... }
```

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

a) the **version** component shall according to [IETF RFC 5652] take the value **v4**;

b) the **kekid** component shall hold an instance of the **KEKIdentifier** data type with the following components:

   – the **keyIdentifier** component shall hold identifier for CEK retained from a previous exchanged as specified in clause B.3.4;

   – the other components are not necessary;

c) the **keyEncryptionAlgorithm** component shall hold the wrapping algorithm as specified in clause B.3.3, item d). It is recommended to use the same wrapping algorithm for all content instances for a particular telebiometrics session.

B.5 Attributes

The following attributes type are defined by [IETF RFC 5652]. Instances of these attribute types are intended to be included as signed attributes.

```
contentType ATTRIBUTE ::= {
  WITH SYNTAX             CONTENT-TYPE.&id({envelopedData, ...})
  EQUALITY MATCHING RULE objectIdentifierMatch
  SINGLE VALUE            TRUE
  ID                      id-contentType }

messageDigest ATTRIBUTE ::= {
  WITH SYNTAX             OCTET STRING
  EQUALITY MATCHING RULE octetStringMatch
  SINGLE VALUE            TRUE
  ID                      id-messageDigest }
The following attributes type are defined by [IETF RFC 3185]. Instances of these
attribute types may be included as unsigned attributes.

aa-CEKReference ATTRIBUTE ::= {
  WITH SYNTAX             CEKReference
  EQUALITY MATCHING RULE octetStringMatch
  SINGLE VALUE            TRUE
  ID                      id-aa-CEKReference }

CEKReference ::= OCTET STRING

aa-CEKMaxDecrypts ATTRIBUTE ::= {
  WITH SYNTAX             CEKMaxDecrypts
  EQUALITY MATCHING RULE integerMatch
  SINGLE VALUE            TRUE
  ID                      id-aa-CEKMaxDecrypts }

CEKMaxDecrypts ::= INTEGER

aa-KEKDerivationAlg ATTRIBUTE ::= {
  WITH SYNTAX             KEKDerivationAlgorithm
  EQUALITY MATCHING RULE integerMatch
  SINGLE VALUE            TRUE
  ID                      id-aa-KEKDerivationAlg }

KEKDerivationAlgorithm ::= SEQUENCE {
  kekAlg       AlgorithmIdentifier,
  pbkdf2Param  PBKDF2-params }

PBKDF2-params ::= SEQUENCE {
  salt CHOICE {
    specified   OCTET STRING,
--  otherSource AlgorithmIdentifier {{PBKDF2-SaltSources}}
    ... },
```

```
  iterationCount INTEGER (1..MAX),
  keyLength      INTEGER (1..MAX) OPTIONAL,
  prf            AlgorithmIdentifier {{PBKDF2-PRFs}},
  ... }

PBKDF2-PRFs ALGORITHM ::= {...}

PBKDF2-params ::= SEQUENCE {
  salt CHOICE {
    specified   OCTET STRING,
    otherSource AlgorithmIdentifier {{PBKDF2-SaltSources}} },
  iterationCount INTEGER (1..MAX),
  keyLength      INTEGER (1..MAX) OPTIONAL,
  prf            AlgorithmIdentifier {{PBKDF2-PRFs}} DEFAULT algid-hmacWithSHA1
}


id-pkcs OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) usa(840) rsadsi(113549) pkcs(1) }


id-pkcs-9 OBJECT IDENTIFIER ::= { id-pkcs pkcs-9(9) }

id-aa OBJECT IDENTIFIER ::= { id-pkcs-9 smime(16) attributes(2) }

id-contentType         OBJECT IDENTIFIER ::= { id-pkcs-9 3 }
id-messageDigest       OBJECT IDENTIFIER ::= { id-pkcs-9 4 }
id-aa-CEKReference     OBJECT IDENTIFIER ::= { id aa 30 }
id-aa-CEKMaxDecrypts   OBJECT IDENTIFIER ::= { id aa 31 }
id-aa-KEKDerivationAlg OBJECT IDENTIFIER ::= { id aa 32 }
```

B.6 Cryptographic message syntax error codes

[b-IETF RFC 7191] provides a list of CMS error codes for all possible uses of CMS. Listed below is a subset of those error codes relevant for telebiometrics. For the description of the error codes, please consult [b-IETF RFC 7191].

When [b-IETF RFC 7191] refers to a certificate, it is a reference to the public-key certificate used for the CMS defined content types.

```
CmsErrorCode ::= ENUMERATED {
  decodeFailure                    (1),
  badContentInfo                   (2),
  badSignedData                    (3),
  badEncapContent                  (4),
  badCertificate                   (5),
  badSignerInfo                    (6),
  badSignedAttrs                   (7),
  badUnsignedAttrs                 (8),
  missingContent                   (9),
  noTrustAnchor                   (10),
  notAuthorized                   (11),
  badDigestAlgorithm              (12),
  badSignatureAlgorithm           (13),
  unsupportedKeySize              (14),
  unsupportedParameters           (15),
  signatureFailure                (16),
  incorrectTarget                 (23),
  missingSignature                (29),
  versionNumberMismatch           (31),
  revokedCertificate              (33),
  badEncryptedData                (62),
  badEnvelopedData                (63),
```

```
badKeyAgreeRecipientInfo         (66),
badKEKRecipientInfo              (67),
badEncryptContent                (68),
badEncryptAlgorithm              (69),
missingCiphertext                (70),
decryptFailure                   (71),
badMACAlgorithm                  (72),
badAuthAttrs                     (73),
badUnauthAttrs                   (74),
invalidMAC                       (75),
mismatchedDigestAlg              (76),
missingCertificate               (77),
tooManySigners                   (78),
missingSignedAttributes          (79),
derEncodingNotUsed               (80),
invalidAttributeLocation         (82),
badAttributes                    (85),
noMatchingRecipientInfo          (91),
unsupportedKeyWrapAlgorithm      (92),
badKeyTransRecipientInfo         (93),
other                           (127) }
```

Annex C

Formal specification of the privilege assertion and assignment protocols

(This annex forms an integral part of this Recommendation.)

```
Pbact-access { joint-iso-itu-t(2) telebiometrics(42) e-health-
protocol(3)telehealthprot(10)
   Part0 (0) modules(0) pbact-access(62) version1(1) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All

IMPORTS

  -- from Rec. ITU-T X.501 | ISO/IEC 9594-2

  ATTRIBUTE, Attribute{}, AttributeType, AttributeTypeAndValue,
  AttributeValueAssertion, DistinguishedName, OBJECT-CLASS, SupportedAttributes
    FROM InformationFramework {joint-iso-itu-t ds(5) module(1)
         informationFramework(1) 8}

  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8

  AttributeCertificate
    FROM AttributeCertificateDefinitions {joint-iso-itu-t ds(5) module(1)
         attributeCertificateDefinitions(32) 8}

  CmsErrorCode, CONTENT-TYPE
    FROM CmsTelebiometric { joint-iso-itu-t(2) telebiometrics(42)
telehealthprot(310)
      part0(0)
      modules(0) cmsProfile(1) version1(1) } ;

accessService ATTRIBUTE ::= {
  WITH SYNTAX  AccessService
  ID           id-at-accessService }

AccessService ::= SEQUENCE {
  serviceId          OBJECT IDENTIFIER,
  objectDef          SEQUENCE SIZE (1..MAX) OF ObjectSel,
  ... }

ObjectSel ::= SEQUENCE {
  objecClass         OBJECT-CLASS.&id,
  objSelect          CHOICE {
    allObj        [0]  TargetSelect,
    objectNames   [1]  SEQUENCE SIZE (1..MAX) OF SEQUENCE {
      object             CHOICE {
        names         [1]  SEQUENCE SIZE (1..MAX) OF DistinguishedName,
        subtree       [2]  DistinguishedName,
        ... },
      select           TargetSelect,
      ... },
    ... },
  ... }

TargetSelect ::= SEQUENCE {
  objOper   ObjectOperations OPTIONAL,
  attrSel   AttributeSel     OPTIONAL,
  ... }
  (WITH COMPONENTS {..., objOper  PRESENT } |
```

```
   WITH COMPONENTS {..., attrSel  PRESENT } )


AttributeSel ::= SEQUENCE {
  attSelect           CHOICE {
    allAttr        [0]  SEQUENCE {
      attrOper1      [0]  AttributeOperations OPTIONAL,
      ... },
    attributes     [1]  SEQUENCE SIZE (1..MAX) OF SEQUENCE {
      select              SEQUENCE SIZE (1..MAX) OF ATTRIBUTE.&id,
      attrOper2      [0]  AttributeOperations OPTIONAL,
      ... },
    ... },
  ... }

ObjectOperations ::= BIT STRING {
  read            (0),
  add             (1),
  modify          (2),
  delete          (3),
  rename          (4),
  discloseOnError (5) }

AttributeOperations ::= BIT STRING {
  read             (0),
  compare          (1),
  add              (2),
  modify           (3),
  delete           (4),
  deleteValue      (5),
  replaceAttribute (6),
  discloseOnError  (7) }

PbactContentTypes CONTENT-TYPE ::= {
    privAssignRequest |
    privAssignResult |
    readRequest |
    readResult |
    compareRequest |
    compareResult |
    addRequest |
    addResult |
    deleteRequest |
    deleteResult |
    modifyRequest |
    modifyResult |
    renameRequest |
    renameResult,
    ... }

CommonReqComp ::= SEQUENCE {
  attrCerts [31]  AttributeCertificates OPTIONAL,
  serviceId [30]  OBJECT IDENTIFIER,
  invokId   [29]  INTEGER,
  ... }

AttributeCertificates ::= SEQUENCE SIZE (1..MAX) OF AttributeCertificate

readRequest CONTENT-TYPE ::= {
              ReadRequest
IDENTIFIED BY id-readRequest }

ReadRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
```

```
  object    [1] DistinguishedName,
  selection [2] InformationSelection,
  ... }

readResult CONTENT-TYPE ::= {
              ReadResult
IDENTIFIED BY id-readResult }


ReadResult ::= SEQUENCE {
  object    DistinguishedName,
  result    CHOICE {
    success   [0] ObjectInformation,
    failure   [1] AccessdErr,
    ... },
  ... }

compareRequest CONTENT-TYPE ::= {
              CompareRequest
IDENTIFIED BY id-compareRequest }

CompareRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object    [1] DistinguishedName,
  purported [2] AttributeValueAssertion,
  ... }

compareResult CONTENT-TYPE ::= {
              CompareResult
IDENTIFIED BY id-compareResult }

CompareResult ::= SEQUENCE {
  object    DistinguishedName,
  result    CHOICE {
    success   [0] CompareOK,
    failure   [1] AccessdErr,
    ... },
  ... }

CompareOK ::= SEQUENCE {
  matched        [0] BOOLEAN,
  matchedSubtype [1] BOOLEAN DEFAULT FALSE,
  ... }

addRequest CONTENT-TYPE ::= {
              AddRequest
IDENTIFIED BY id-addRequest }

AddRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object    [1] DistinguishedName,
  attr      [2] SEQUENCE SIZE (1..MAX) OF Attribute {{SupportedAttributes}}
                 OPTIONAL,
  ... }

addResult CONTENT-TYPE ::= {
              AddResult
IDENTIFIED BY id-addResult }

AddResult ::= CHOICE {
  success   [0] NULL,
  failure   [1] AccessdErr,
  ... }
```

```
deleteRequest CONTENT-TYPE ::= {
              DeleteRequest
IDENTIFIED BY id-deleteRequest }

DeleteRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object        DistinguishedName,
  ... }

deleteResult CONTENT-TYPE ::= {
              DeleteResult
IDENTIFIED BY id-deleteResult }

DeleteResult ::= CHOICE {
  success   [0] NULL,
  failure   [1] AccessdErr,
  ... }

modifyRequest CONTENT-TYPE ::= {
              ModifyRequest
IDENTIFIED BY id-modifyRequest }

ModifyRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object        DistinguishedName,
  changes       SEQUENCE SIZE (1..MAX) OF ObjectModification,
  select        InformationSelection,
  ... }

ObjectModification ::= CHOICE {
  addAttribute     [0]  Attribute{{SupportedAttributes}},
  deleteAttribute  [1]  AttributeType,
  addValues        [2]  Attribute{{SupportedAttributes}},
  deleteValues     [3]  Attribute{{SupportedAttributes}},
  replaceAttribute [4]  Attribute{{SupportedAttributes}},
  ... }

modifyResult CONTENT-TYPE ::= {
              ModifyResult
IDENTIFIED BY id-modifyResult }

ModifyResult ::= SEQUENCE {
  result    CHOICE {
    success   [0] ObjectInformation,
    failure   [1] AccessdErr,
    ... },
  ... }

renameRequest CONTENT-TYPE ::= {
              RenameRequest
IDENTIFIED BY id-renameRequest }

RenameRequest ::= SEQUENCE {
  COMPONENTS OF CommonReqComp,
  object        DistinguishedName,
  new           DistinguishedName,
  ... }

renameResult CONTENT-TYPE ::= {
              RenameResult
IDENTIFIED BY id-renameResult }

RenameResult ::= SEQUENCE {
  result    CHOICE {
```

```
    success   [0] NULL,
    failure   [1] AccessdErr,
    ... },
  ... }

AccessdErr ::= CHOICE {
  cmsErr     [0] CmsErrorCode,
  pbactErr   [1] PbactErr,
  ... }

InformationSelection ::= SEQUENCE {
  attributes      CHOICE {
    allAttributes   [0]  NULL,
    select          [1]  SEQUENCE SIZE (1..MAX) OF ATTRIBUTE.&id,
    ... },
  infoTypes        ENUMERATED {
    attributeTypesOnly     (0),
    attributeTypeAndValue  (1),
    ... },
  ... }

ObjectInformation ::= SEQUENCE {
  name   DistinguishedName,
  info   SET SIZE (1..MAX) OF Attribute {{SupportedAttributes}},
  ... }

PbactErr ::= ENUMERATED {
  noSuchService,
  invalidOperationForService,
  insufficientAccessRight,
  noSuchObject,
  noSuchAttribute,
  noSuchAttributeValue,
  objectAlreadyExists,
  attributeAlreadyExists,
  attributeValueAlreadyExists,
  noInformation,
  ... }

privAssignRequest CONTENT-TYPE ::= {
              PrivAssignRequest
IDENTIFIED BY id-privAssignRequest }

PrivAssignRequest ::= SEQUENCE {
  attrCerts [1]  AttributeCertificates OPTIONAL,
  ... }

privAssignResult CONTENT-TYPE ::= {
              PrivAssignResult
IDENTIFIED BY id-privAssignResult }

PrivAssignResult ::= SEQUENCE {
  result  CHOICE {
    success  NULL,
    failure  PrivAssignErr },
  ... }

PrivAssignErr ::= CHOICE {
  cmsErr     [0] CmsErrorCode,
  assignErr  [1] AssignErr,
  ... }

AssignErr ::= ENUMERATED {
  invalidAttributeCertificate (0),
```

```
  ... }

-- object identifier allocations

-- top tree

id-pbacttelebio          --  OBJECT IDENTIFIER ::=
    { joint-iso-itu-t(2) telebiometrics(42) e-health-protocol(3) pbact(20) }
id-pbactmoduletelehealthprot            OBJECT IDENTIFIER ::= { id-pbact
moduletelebio telehealthprot(10) }
id-pbactContx1080-0          OBJECT IDENTIFIER ::= { id-pbact cmsCont(1thprot
part0 (0) }
id-pbactPrivAttrx1080-0-module    -- OBJECT IDENTIFIER ::= { id-pbact x1080-0
module(0) }
id-x1080-0-Cont      -- OBJECT IDENTIFIER ::= { id-x1080-0 cmsCont(1) }
id-x1080-0-attr      -- OBJECT IDENTIFIER ::= { id-x1080-0 prAttr(2) }

-- Content types

id-privAssignRequest OBJECT IDENTIFIER ::=
    { id-pbactContx1080-0-Cont privAssignRequest(1) }
id-privAssignResult  OBJECT IDENTIFIER ::=
    { id-pbactContx1080-0-Cont privAssignResult(2) }
id-readRequest       OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
readRequest(3) }
id-readResult        OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
readResult(4) }
id-compareRequest    OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
compareRequest(5) }
id-compareResult     OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
compareResult(6) }
id-addRequest        OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
addRequest(7) }
id-addResult         OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
addResult(8) }
id-deleteRequest     OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
deleteRequest(9) }
id-deleteResult      OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
deleteResult(10) }
id-modifyRequest     OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
modifyRequest(11) }
id-modifyResult      OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
modifyResult(12) }
id-renameRequest     OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
renameRequest(13) }
id-renameResult      OBJECT IDENTIFIER ::= { id-pbactContx1080-0-Cont
renameResult(14) }


-- Axttribute types for carrying privilege definitions

id-at-accessService OBJECT IDENTIFIER ::= { id-pbactPrivAttrx1080-0-attr 1 }

END
```

Appendix I

Informal specification for the cryptographic message syntax profile

(This appendix does not form an integral part of this Recommendation.)

An implementation only supporting the `CmsTelebiometric` module is not conformant with the IETF CMS specification and shall not be seen as an implementation specification. It is provided here for information and for consistency checking.

```
CmsTelebiometric { joint-iso-itu-t(2) telebiometrics(42) thprot(130)
  part0(0)
  modules(0) cmsProfile(1) version1(1) }
DEFINITIONS ::=
BEGIN

-- EXPORTS All

IMPORTS

  -- from Rec. ITU-T X.501 | ISO/IEC 9594-2

      ATTRIBUTE, Attribute{}, DistinguishedName, objectIdentifierMatch
    FROM InformationFramework {joint-iso-itu-t ds(5) module(1)
informationFramework(1) 8}

  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8

  ALGORITHM, AlgorithmIdentifier, Certificate, CertificateSerialNumber
    FROM AuthenticationFramework {joint-iso-itu-t ds(5) module(1)
authenticationFramework(7) 8}

  -- from Rec. ITU-T X.520 | ISO/IEC 9594-6

  integerMatch, octetStringMatch
    FROM SelectedAttributeTypes {joint-iso-itu-t ds(5) module(1)
selectedAttributeTypes(5) 8}  ;

CONTENT-TYPE ::= TYPE-IDENTIFIER

ContentType ::= CONTENT-TYPE.&id

ContentInfo ::= SEQUENCE {
  contentType  CONTENT-TYPE.&id ({TelebSupportedcontentTypes}),
  content      CONTENT-TYPE.&Type
          ({TelebSupportedcontentTypes}{@contentType})OPTIONAL,
  ... }

TelebSupportedcontentTypes CONTENT-TYPE ::=
  { signedData | envelopedData | ct-authEnvelopedData, ...}

CMSVersion ::= INTEGER{ v0(0), v1(1), v2(2), v3(3), v4(4), v5(5) }

Attributes { ATTRIBUTE:AttrList } ::=
             SET SIZE (1..MAX) OF Attribute {{ AttrList }}

signedData CONTENT-TYPE ::= {
                SignedData
  IDENTIFIED BY id-signedData }

SignedData ::= SEQUENCE  {
  version               CMSVersion (v3),
```

```
  digestAlgorithms      SET (SIZE (1)) OF AlgorithmIdentifier
                          {{Teleb-Hash-Algorithms}},
  encapContentInfo      EncapsulatedContentInfo,
  certificates     [0]  IMPLICIT SET (SIZE (1..MAX)) OF Certificate OPTIONAL,
--crls             [1]  IMPLICIT RevocationInfoChoices OPTIONAL,
  signerInfos           SignerInfos,
  ... }

Teleb-Hash-Algorithms ALGORITHM ::= {...}

EncapsulatedContentInfo ::= SEQUENCE {
  eContentType     CONTENT-TYPE.&id({IncludedContent}),
  eContent     [0] EXPLICIT OCTET STRING
    (CONTAINING CONTENT-TYPE.&Type({IncludedContent}
     {@eContentType})) OPTIONAL }

IncludedContent CONTENT-TYPE ::= {envelopedData, ...}

SignerInfos ::= SET (SIZE (1)) OF SignerInfo

SignerInfo ::= SEQUENCE {
  version                 CMSVersion (v1),
  sid                     SignerIdentifier,
  digestAlgorithm         AlgorithmIdentifier {{Teleb-Hash-Algorithms}},
  signedAttrs        [0]  IMPLICIT Attributes{{SignedAttributes}} OPTIONAL,
  signatureAlgorithm      AlgorithmIdentifier {{Teleb-Signature-Algorithms}},
  signature               SignatureValue,
  unsignedAttrs      [1]  IMPLICIT Attributes {{UnsignedAttributes}} OPTIONAL,
  ... }

SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber    IssuerAndSerialNumber,
--subjectKeyIdentifier [0] SubjectKeyIdentifier,
  ...}

IssuerAndSerialNumber ::= SEQUENCE {
  issuer       DistinguishedName,
  serialNumber CertificateSerialNumber }

SignedAttributes ATTRIBUTE ::= { contentType | messageDigest, ... }

Teleb-Signature-Algorithms ALGORITHM ::= {...}

SignatureValue ::= OCTET STRING

UnsignedAttributes ATTRIBUTE ::= {...}

envelopedData CONTENT-TYPE ::= {
                EnvelopedData
  IDENTIFIED BY id-envelopedData }

EnvelopedData ::= SEQUENCE {
  version                   CMSVersion(v0 | v2),
--originatorInfo         [0] IMPLICIT OriginatorInfo OPTIONAL,
  recipientInfos            RecipientInfos,
  encryptedContentInfo      EncryptedContentInfo,
  ...,
  [[2: unprotectedAttrs  [1] IMPLICIT Attributes
       {{UnprotectedAttributes}} OPTIONAL ]] }

RecipientInfos ::= SET SIZE (1) OF RecipientInfo

UnprotectedAttributes ATTRIBUTE ::=
  { aa-CEKReference | aa-CEKMaxDecrypts | aa-KEKDerivationAlg }
```

```
RecipientInfo ::= CHOICE {
--ktri      KeyTransRecipientInfo,
  kari  [1] KeyAgreeRecipientInfo,
  kekri [2] KEKRecipientInfo,
--pwri  [3] PasswordRecipientinfo,
--ori   [4] OtherRecipientInfo,
  ... }

KeyAgreeRecipientInfo ::= SEQUENCE {
  version                 CMSVersion (v3),
  originator          [0] EXPLICIT OriginatorIdentifierOrKey,
  ukm                 [1] EXPLICIT UserKeyingMaterial OPTIONAL,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  recipientEncryptedKeys RecipientEncryptedKeys,
  ... }

OriginatorIdentifierOrKey ::= CHOICE {
  issuerAndSerialNumber    IssuerAndSerialNumber,
--subjectKeyIdentifier [0] SubjectKeyIdentifier,
  originatorKey        [1] OriginatorPublicKey,
  ... }

OriginatorPublicKey ::= SEQUENCE {
  algorithm AlgorithmIdentifier {{SupportedDHPublicKeyAlgorithms}},
  publicKey BIT STRING,
  ... }

SupportedDHPublicKeyAlgorithms ALGORITHM ::= {...}

UserKeyingMaterial ::= OCTET STRING (SIZE (64))

KeyEncryptionAlgorithmIdentifier ::=
   AlgorithmIdentifier{{SupportedKeyIncryptAlgorithms}}

SupportedKeyIncryptAlgorithms ALGORITHM ::= {...}

RecipientEncryptedKeys ::= SEQUENCE (SIZE (1)) OF RecipientEncryptedKey

RecipientEncryptedKey ::= SEQUENCE {
  rid          KeyAgreeRecipientIdentifier,
  encryptedKey EncryptedKey }

KeyAgreeRecipientIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
--rKeyId            [0] IMPLICIT RecipientKeyIdentifier,
  ... }

EncryptedKey ::= OCTET STRING

EncryptedContentInfo ::= SEQUENCE {
  contentType                CONTENT-TYPE.&id ({EncryptedContentSet}),
  contentEncryptionAlgorithm SEQUENCE {
    algorithm                  ALGORITHM.&id ({SymmetricEncryptionAlgorithms}),
    parameter                  ALGORITHM.&Type
             ({SymmetricEncryptionAlgorithms}{@.algorithm})} OPTIONAL,
  encryptedContent       [0] IMPLICIT EncryptedContent OPTIONAL,
  ... }

EncryptedContentSet CONTENT-TYPE ::= {...}

SymmetricEncryptionAlgorithms ALGORITHM ::= {...}

EncryptedContent ::= OCTET STRING
```

```
ct-authEnvelopedData CONTENT-TYPE ::= {
                AuthEnvelopedData
  IDENTIFIED BY id-ct-authEnvelopedData }

AuthEnvelopedData ::= SEQUENCE {
  version                   CMSVersion (v0),
--originatorInfo       [0] IMPLICIT OriginatorInfo OPTIONAL,
  recipientInfos            RecipientInfos,
  authEncryptedContentInfo EncryptedContentInfo,
  authAttrs             [1] IMPLICIT Attributes {{AuthAttributes}} OPTIONAL,
  mac                       MessageAuthenticationCode,
  unauthAttrs           [2] IMPLICIT Attributes {{UnauthAttributes}} OPTIONAL }

AuthAttributes ATTRIBUTE ::= {...}

MessageAuthenticationCode ::= OCTET STRING

UnauthAttributes ATTRIBUTE ::=
  { aa-CEKReference | aa-CEKMaxDecrypts | aa-KEKDerivationAlg }

KEKRecipientInfo ::= SEQUENCE {
  version                CMSVersion (v4),
  kekid                  KEKIdentifier,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey           EncryptedKey }

KEKIdentifier ::= SEQUENCE {
  keyIdentifier OCTET STRING,
--date          GeneralizedTime OPTIONAL,
--other         OtherKeyAttribute OPTIONAL,
  ... }

contentType ATTRIBUTE ::= {
  WITH SYNTAX            CONTENT-TYPE.&id({envelopedData, ...})
  EQUALITY MATCHING RULE objectIdentifierMatch
  SINGLE VALUE           TRUE
  ID                     id-contentType }

messageDigest ATTRIBUTE ::= {
  WITH SYNTAX            OCTET STRING
  EQUALITY MATCHING RULE octetStringMatch
  SINGLE VALUE           TRUE
  ID                     id-messageDigest }

aa-CEKReference ATTRIBUTE ::= {
  WITH SYNTAX            CEKReference
  EQUALITY MATCHING RULE octetStringMatch
  SINGLE VALUE           TRUE
  ID                     id-aa-CEKReference }

CEKReference ::= OCTET STRING

aa-CEKMaxDecrypts ATTRIBUTE ::= {
  WITH SYNTAX            CEKMaxDecrypts
  EQUALITY MATCHING RULE integerMatch
  SINGLE VALUE           TRUE
  ID                     id-aa-CEKReference }

CEKMaxDecrypts ::= INTEGER

aa-KEKDerivationAlg ATTRIBUTE ::= {
  WITH SYNTAX            KEKDerivationAlgorithm
  EQUALITY MATCHING RULE integerMatch
```

```
  SINGLE VALUE           TRUE
  ID                     id-aa-KEKDerivationAlg }

KEKDerivationAlgorithm ::= SEQUENCE {
  kekAlg       AlgorithmIdentifier {{SupportedKeyIncryptAlgorithms}},
  pbkdf2Param  PBKDF2-params }

PBKDF2-params ::= SEQUENCE {
  salt CHOICE {
    specified   OCTET STRING,
--  otherSource AlgorithmIdentifier {{PBKDF2-SaltSources}}
    ... },
  iterationCount INTEGER (1..MAX),
  keyLength      INTEGER (1..MAX) OPTIONAL,
  prf            AlgorithmIdentifier {{PBKDF2-PRFs}},
  ... }

PBKDF2-PRFs ALGORITHM ::= {...}

id-pkcs OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) usa(840) rsadsi(113549) pkcs(1) }

id-pkcs-9 OBJECT IDENTIFIER ::= { id-pkcs pkcs-9(9) }

id-ct OBJECT IDENTIFIER ::= { id-pkcs-9 smime(16) ct(1) }
id-aa OBJECT IDENTIFIER ::= { id-pkcs-9 smime(16) attributes(2) }

id-contentType         OBJECT IDENTIFIER ::= { id-pkcs-9 3 }
id-messageDigest       OBJECT IDENTIFIER ::= { id-pkcs-9 4 }
id-aa-CEKReference     OBJECT IDENTIFIER ::= { id-aa 30 }
id-aa-CEKMaxDecrypts   OBJECT IDENTIFIER ::= { id-aa 31 }
id-aa-KEKDerivationAlg OBJECT IDENTIFIER ::= { id-aa 32 }

id-signedData OBJECT IDENTIFIER ::= {iso(1) member-body(2)
us(840)rsadsi(113549) pkcs(1) pkcs7(7) 2}

id-envelopedData OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
rsadsi(113549) pkcs(1) pkcs7(7) 3}

id-ct-authEnvelopedData OBJECT IDENTIFIER ::= { id-ct 23 }

END -- CmsTelebiometric
```

Bibliography

[b-ITU-T X.841] Recommendation ITU-T X.841 (2000) | ISO/IEC 15816:2002, *Information technology – Security techniques – Security information objects for access control*.

[b-IEC 62351-8] IEC TS 62351-8:2011, *Power systems management and associated information exchange– Data and communications security – Part 8: Role-based access control*.

[b-NIST 800-56A] NIST Special Publication 800-56A, Revision 2 (2013), *Recommendation for Pair-Wise Key-Establishment. Schemes Using Discrete Logarithm Cryptography*.

[b-NIST 800-162] NIST Special Publication 800-162 (2014), *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*.

[b-IETF RFC 5480] IETF RFC 5480 (2009), *Elliptic Curve Cryptography Subject Public Key Information*.

[b-IETF RFC 7191] IETF RFC 7191 (2014), *Cryptographic Message Syntax (CMS) – Key Package Receipt and Error Content Types*.

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, 2018