Union internationale des télécommunications

UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

X.1080.0

(03/2017)

SÉRIE X: RÉSEAUX DE DONNÉES, COMMUNICATION ENTRE SYSTÈMES OUVERTS ET SÉCURITÉ

Sécurité de l'information et des réseaux – Télébiométrie

Contrôle d'accès pour la protection des données télébiométriques

Recommandation UIT-T X.1080.0

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX DE DONNÉES, COMMUNICATION ENTRE SYSTÈMES OUVERTS ET SÉCURITÉ

RÉSEAUX PUBLICS DE DONNÉES	X.1–X.199
INTERCONNEXION DES SYSTÈMES OUVERTS	X.200–X.299
INTERFONCTIONNEMENT DES RÉSEAUX	X.300–X.399
SYSTÈMES DE MESSAGERIE	X.400–X.499
ANNUAIRE	X.500–X.599
RÉSEAUTAGE OSI ET ASPECTS SYSTÈMES	X.600–X.699
GESTION OSI	X.700–X.799
SÉCURITÉ	X.800–X.849
APPLICATIONS OSI	X.850–X.899
TRAITEMENT RÉPARTI OUVERT	X.900–X.999
SÉCURITÉ DE L'INFORMATION ET DES RÉSEAUX	
Aspects généraux de la sécurité	X.1000–X.1029
Sécurité des réseaux	X.1030–X.1049
Gestion de la sécurité	X.1050–X.1069
Télébiométrie	**X.1080–X.1099**
APPLICATIONS ET SERVICES SÉCURISÉS	
Sécurité en multidiffusion	X.1100–X.1109
Sécurité des réseaux domestiques	X.1110–X.1119
Sécurité des télécommunications mobiles	X.1120–X.1139
Sécurité de la toile	X.1140–X.1149
Protocoles de sécurité	X.1150–X.1159
Sécurité d'homologue à homologue	X.1160–X.1169
Sécurité des identificateurs en réseau	X.1170–X.1179
Sécurité de la télévision par réseau IP	X.1180–X.1199
SÉCURITÉ DU CYBERESPACE	
Cybersécurité	X.1200–X.1229
Lutte contre le pollupostage	X.1230–X.1249
Gestion des identités	X.1250–X.1279
APPLICATIONS ET SERVICES SÉCURISÉS	
Communications d'urgence	X.1300–X.1309
Sécurité des réseaux de capteurs ubiquitaires	X.1310–X.1339
Recommandations relatives aux infrastructures de clé publique	X.1340–X.1349
Sécurité de l'Internet des objets (IoT)	X.1360–X.1369
Sécurité des systèmes de transport intelligents	X.1370–X.1379
ECHANGE D'INFORMATIONS SUR LA CYBERSÉCURITÉ	
Aperçu général de la cybersécurité	X.1500–X.1519
Echange concernant les vulnérabilités/les états	X.1520–X.1539
Echange concernant les événements/les incidents/l'heuristique	X.1540–X.1549
Echange de politiques	X.1550–X.1559
Heuristique et demande d'informations	X.1560–X.1569
Identification et découverte	X.1570–X.1579
Echange garanti	X.1580–X.1589
SÉCURITÉ DE L'INFORMATIQUE EN NUAGE	
Aperçu de la sécurité de l'informatique en nuage	X.1600–X.1601
Conception de la sécurité de l'informatique en nuage	X.1602–X.1639
Bonnes pratiques et lignes directrices concernant la sécurité de l'informatique en nuage	X.1640–X.1659
Mise en oeuvre de la sécurité de l'informatique en nuage	X.1660–X.1679
Sécurité de l'informatique en nuage (autres)	X.1680–X.1699

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

Recommandation UIT-T X.1080.0

Contrôle d'accès pour la protection des données télébiométriques

Résumé

La Recommandation UIT-T X.1080.0 spécifie comment protéger les informations télébiométriques contre l'accès non autorisé. Elle adopte un point de vue orienté services, selon lequel seules les informations nécessaires pour un but particulier sont fournies, c'est-à-dire que l'accès est accordé non seulement en fonction du *droit de savoir*, mais aussi en fonction du *besoin de savoir*.

Cette Recommandation définit essentiellement une spécification d'attribut incluse dans un certificat d'attribut ou de clé publique qui spécifie en détail les privilèges dont une entité particulière dispose pour un ou plusieurs types de service.

La sécurité est assurée grâce à l'utilisation d'un profil de la syntaxe de message cryptographique (CMS). Le profil CMS permet d'assurer l'authentification, l'intégrité et, si nécessaire, la confidentialité (chiffrement).

Il est destiné à fournir un appui en matière de sécurité pour les spécifications télébiométriques en général. Ce profil repose sur l'hypothèse et la condition du déploiement correct d'une infrastructure de clé publique (PKI).

Cette Recommandation dépend aussi du déploiement d'une infrastructure de gestion des privilèges (PMI).

Historique

Edition	Recommandation	Approbation	Commission d'études	Identifiant unique*
1.0	UIT-T X.1080.0	30-03-2017	17	11.1002/1000/13193

Mots clés

Contrôle d'accès, Diffie-Hellman, PKI, télébiométrie.

---

* Pour accéder à la Recommandation, reporter cet URL http://handle.itu.int/ dans votre navigateur web, suivi de l'identifiant unique. Par exemple, http://handle.itu.int/11.1002/1000/11830-en.

AVANT-PROPOS

L'Union internationale des télécommunications (UIT) est une institution spécialisée des Nations Unies dans le domaine des télécommunications et des technologies de l'information et de la communication (ICT). Le Secteur de la normalisation des télécommunications (UIT-T) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

Le respect de cette Recommandation se fait à titre volontaire. Cependant, il se peut que la Recommandation contienne certaines dispositions obligatoires (pour assurer, par exemple, l'interopérabilité et l'applicabilité) et considère que la Recommandation est respectée lorsque toutes ces dispositions sont observées. Le futur d'obligation et les autres moyens d'expression de l'obligation comme le verbe "devoir" ainsi que leurs formes négatives servent à énoncer des prescriptions. L'utilisation de ces formes ne signifie pas qu'il est obligatoire de respecter la Recommandation.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT n'avait pas été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux développeurs de consulter la base de données des brevets du TSB sous http://www.itu.int/ITU-T/ipr/.

© UIT 2017

Tous droits réservés. Aucune partie de cette publication ne peut être reproduite, par quelque procédé que ce soit, sans l'accord écrit préalable de l'UIT.

Table des matières

Introduction

Le recueil de données télébiométriques personnelles comporte un risque d'atteinte à la vie privée.

Diverses raisons peuvent motiver la protection de ce type d'information, par exemple parce qu'il s'agit d'informations donnant accès à une société ou à une organisation ou d'informations sensibles, qui sont soumises à une restriction de diffusion.

La protection contre la divulgation non désirée de données télébiométriques présente deux grandes composantes:

– protection des données pendant la transmission, en général par chiffrement, et protection des données mémorisées;

– contrôle d'accès des données mémorisées.

Si les systèmes télébiométriques doivent présenter un haut degré de sécurité quant à la confidentialité (chiffrement), à l'authentification, à l'intégrité, à la protection physique, à l'utilisation de pare-feu, aux logiciels antivirus, etc., il est également nécessaire de mettre en place un système perfectionné de contrôle d'accès aux informations mémorisées, en particulier aux données personnelles. Ce dernier point est particulièrement important dans le cas des systèmes télébiométriques.

Les systèmes généraux de contrôle d'accès ne sont pas suffisants, car ils s'appuient essentiellement sur le *droit* (ou le refus) d'accès à l'information, sans tenir §compte, en détail, du *besoin de savoir*, principe selon lequel il faut non seulement avoir le droit de consulter des données, mais aussi apporter la preuve que ces données seront utilisées à des fins légitimes.

Les informations ne devraient être fournies que pour l'usage auxquelles elles sont destinées. Par exemple, les données médicales concernant un patient sont recueillies pour permettre de traiter le patient en question de manière optimale et ne devraient pas être utilisées à d'autres fins, si ce n'est peut-être pour des projets de recherche rigoureusement contrôlés qui nécessitent l'utilisation de certaines informations provenant d'un ensemble particulier de patients. Autrement, les données doivent être protégées contre la pêche aux informations.

On distingue deux grands types de contrôle d'accès: le contrôle d'accès physique et le contrôle d'accès logique. Le contrôle d'accès physique consiste à limiter l'accès aux campus, aux bâtiments, aux salles et aux actifs informatiques physiques. Le contrôle d'accès logique permet de restreindre les connexions aux réseaux informatiques, aux fichiers système et aux données. La présente Recommandation ne traite que du contrôle d'accès logique.

Le contrôle d'accès comprend l'authentification sécurisée des "accesseurs" par le prestataire de services. La présente Recommandation se place dans l'hypothèse suivante: utilisation de signatures numériques et existence d'une infrastructure de clé publique (PKI).

La Recommandation UIT-T X.1080.0 peut être citée dans d'autres spécifications télébiométriques.

L'Annexe A, qui fait partie intégrante de la présente Recommandation, spécifie l'affectation d'identificateurs d'objet utilisés par les Recommandations UIT-T de la série X.1080.

L'Annexe B, qui fait partie intégrante de la présente Recommandation, définit un profil télébiométrique de la syntaxe de message cryptographique (CMS), comme indiqué dans le document IETF RFC 5652, qui est utilisé par la présente Recommandation. Ce profil peut aussi être cité par d'autres spécifications télébiométriques.

L'Annexe C, qui fait partie intégrante de la présente Recommandation, spécifie, de manière formelle, les protocoles de déclaration et d'attribution de privilèges sous la forme d'un module en notation de syntaxe abstraite numéro un (ASN.1).

L'Appendice I, qui ne fait pas partie de la présente Recommandation, spécifie, de manière informelle, le profil CMS sous la forme d'un module ASN.1.

Recommandation UIT-T X.1080.0

Contrôle d'accès pour la protection des données télébiométriques

1 Domaine d'application

La présente Recommandation spécifie comment la vie privée peut être protégée dans un environnement télébiométrique, et ce au moyen d'un contrôle d'accès en télébiométrie (ACT) fondé sur la protection de la vie privée. S'il n'est pas envisageable de recenser ici tous les types d'information possibles, il est néanmoins prévu, dans le cadre de la présente Recommandation, de fournir des outils généraux pour gérer tous les types d'information de façon sécurisée. Il s'agira notamment de définir un protocole d'attribution des privilèges ainsi qu'un protocole d'accès aux informations au moyen d'une déclaration de privilèges. La présente Recommandation fournit des lignes directrices et ne contient pas d'exigences de conformité.

Les éléments suivants sont hors du domaine d'application de la présente Recommandation:

– Protection physique de l'information.

– Accès non autorisé par le personnel d'exploitation qui est chargé d'assurer le bon fonctionnement du système de sécurité et qui a donc la possibilité de le contourner.

2 Références

Les Recommandations UIT-T et autres références suivantes contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions de la présente Recommandation. Au moment de la publication, les éditions indiquées étaient en vigueur. Les Recommandations et autres références étant sujettes à révision, les utilisateurs de la présente Recommandation sont invités à rechercher la possibilité d'appliquer les éditions les plus récentes des Recommandations et autres références énumérées ci-dessous. Une liste des Recommandations UIT-T en vigueur est publiée périodiquement. La référence à un document figurant dans la présente Recommandation ne donne pas à ce document en tant que tel le statut de Recommandation.

[Série UIT-T X.500] Série de Recommandations UIT-T X.5xx (2016) | ISO/CEI 9594-x, *Technologies de l'information – Interconnexion des systèmes ouverts – L'annuaire*.

[UIT-T X.501] Recommandation UIT-T X.501 (2016) | ISO/CEI 9594-2, *Technologies de l'information – Interconnexion des systèmes ouverts – L'annuaire: les modèles*.

[UIT-T X.509] Recommandation UIT-T X.509 (2016) | ISO/CEI 9594-8, *Technologies de l'information – Interconnexion des systèmes ouverts – L'annuaire: cadre général des certificats de clé publique et d'attribut*.

[UIT-T X.520] Recommandation UIT-T X.520 (2016) | ISO/CEI 9594-6, *Technologies de l'information – Interconnexion des systèmes ouverts – L'annuaire: types d'attributs sélectionnés*.

[UIT-T X.521] Recommandation UIT-T X.521 (2016) | ISO/CEI 9594-7, *Technologies de l'information – Interconnexion des systèmes ouverts – L'annuaire: classes d'objets sélectionnées*.

[UIT-T X.680] Recommandation UIT-T X.680 (2015) | ISO/CEI 8824-1, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base*.

[UIT-T X.681] Recommandation UIT-T X.681 (2015) | ISO/CEI 8824-2, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des objets informationnels*.

[UIT-T X.682] Recommandation UIT-T X.682 (2015) | ISO/CEI 8824-3, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des contraintes*.

[UIT-T X.683] Recommandation UIT-T X.683 (2015) | ISO/CEI 8824-4, *Technologies de l'information – Syntaxe abstraite numéro un: paramétrage des spécifications de la notation de syntaxe abstraite numéro un.*

[UIT-T X.690] Recommandation UIT-T X.690 (2015) | ISO/CEI 8825-1, *Technologies de l'information – Notation de syntaxe abstraite numéro un: paramétrage des spécifications de la notation de syntaxe abstraite numéro un*.

[UIT-T X.1080.1] Recommandation UIT-T X.1080.1 (2011), *Cybersanté et systèmes mondiaux de télémédecine – Protocole générique de télécommunication*.

[UIT-T X.1081] Recommandation UIT-T X.1081 (2011), *Le modèle télébiométrique multimodal – Cadre général pour la spécification des aspects de sécurité et d'innocuité de la télébiométrie*.

[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 Définitions

3.1 Termes définis ailleurs

La présente Recommandation utilise les termes suivants définis ailleurs:

3.1.1 certificat d'attribut [UIT-T X.509]: structure de données, portant la signature numérique d'une autorité d'attribut, qui lie certaines valeurs d'attribut à des informations d'identification concernant son détenteur.

3.1.2 type d'attribut [UIT-T X.501]: composant d'un attribut qui indique la classe d'informations détenue par cet attribut.

3.1.3 valeur d'attribut [UIT-T X.501]: instance particulière de la classe d'informations indiquée par un type d'attribut.

3.1.4 privilège [UIT-T X.509]: attribut ou propriété attribué par une autorité à une entité.

3.1.5 détenteur de privilège [UIT-T X.509]: entité à laquelle un privilège a été attribué. Un détenteur de privilège peut déclarer son privilège pour un but particulier.

3.1.6 vérificateur de privilège [UIT-T X.509]: entité effectuant la vérification de certificats conformément à une politique de privilège.

3.1.7 source d'autorité (SOA) [UIT-T X.509]: autorité d'attribut à laquelle un vérificateur de privilège fait confiance pour une ressource donnée, en tant qu'autorité ultime pour l'attribution d'un ensemble de privilèges en vue de la déclaration de cette ressource.

3.2 Termes définis dans la présente Recommandation

La présente Recommandation définit les termes suivants:

3.2.1 contrôle d'accès: technique de sécurité utilisée pour réglementer ce qui peut être fait, et par qui, en ce qui concerne des ressources d'information dans un environnement informatique.

3.2.2 service d'accès: service fourni par un prestataire de services pour l'exécution d'une transaction donnée.

3.2.3 accesseur: détenteur d'un privilège qui accède à un service d'accès donné au moyen de son privilège.

3.2.4 attribut: information d'un type particulier associée à un objet. L'information associée à un objet est composée d'attributs.

3.2.5 domaine de protection des données: domaine dans lequel l'information à protéger relève d'un composant de gestion unique.

3.2.6 nom distinctif: nom identifiant un objet qui est unique dans un contexte particulier et qui est formé d'un ou plusieurs composants de nom représentant la position de l'objet dans une hiérarchie d'objets.

3.2.7 objet: personne, service, professionnel ou autre type d'objet à propos duquel on possède des informations et qui est identifiable par un nom distinctif.

3.2.8 classe d'objets: famille identifiée d'entités qui ont certaines caractéristiques en commun.

3.2.9 opération: interaction constituée d'une demande et d'une réponse entre un accesseur et un fournisseur de services dans un but particulier.

3.2.10 spécification: Recommandation UIT-T, norme internationale ou toute spécification élaborée par une organisation de normalisation reconnue.

4 Abréviations et acronymes

La présente Recommandation utilise les abréviations et acronymes suivants:

AA autorité d'attribut

ABAC contrôle d'accès fondé sur des attributs (*attribute based access control*)

ACL liste de contrôles d'accès (*access control list*)

ACT contrôle d'accès pour la télébiométrie (*access control for telebiometrics*)

AES norme de chiffrement perfectionné (*advanced encryption standard*)

ASN.1 notation de syntaxe abstraite numéro un (*abstract syntax notation one*)

BER règles de codage élémentaires (*basic encoding rules*)

CA autorité de certification (*certification authority*)

CEK clé de chiffrement de contenu (*content encryption key*)

CMS syntaxe de message cryptographique (*cryptographic message syntax*)

DER règles de codage distinctives (*distinguished encoding rules*)

DH Diffie-Hellman

ECC cryptographie à courbe elliptique (*elliptic curve cryptography*)

ECDH courbe elliptique Diffie-Hellman (*elliptic curve Diffie-Hellman*)

GCM mode Galois/compteur (*Galois/counter mode*)

KEK clé de chiffrement de clé (*key-encryption key*)

LDAP protocole rapide d'accès à l'annuaire (*lightweight directory access protocol*)

MAC code d'authentification de message (*message authentication code*)

PDU unité de données de protocole (*protocol data unit*)

PKI infrastructure de clé publique (*public key infrastructure*)

PMI infrastructure de gestion de privilège (*privilege management infrastructure*)

SDO organisation de normalisation (*standards developing organization*)

SOA source d'autorité (*source of authority*)

5 Conventions

Dans la présente Recommandation, la notation de syntaxe abstraite numéro un (ASN.1) apparaît en **`caractères gras de la police courier new`**. Lorsque des types et des valeurs ASN.1 sont cités dans le texte normal, ils en sont différenciés par leur présentation en caractères gras de la police "courier new".

Si les éléments d'une liste sont numérotés (par opposition à l'utilisation du caractère "–" ou de lettres), ils doivent être considérés comme les étapes d'une procédure.

6 Concepts et modèles de base

6.1 Protection dans un domaine unique de protection des données

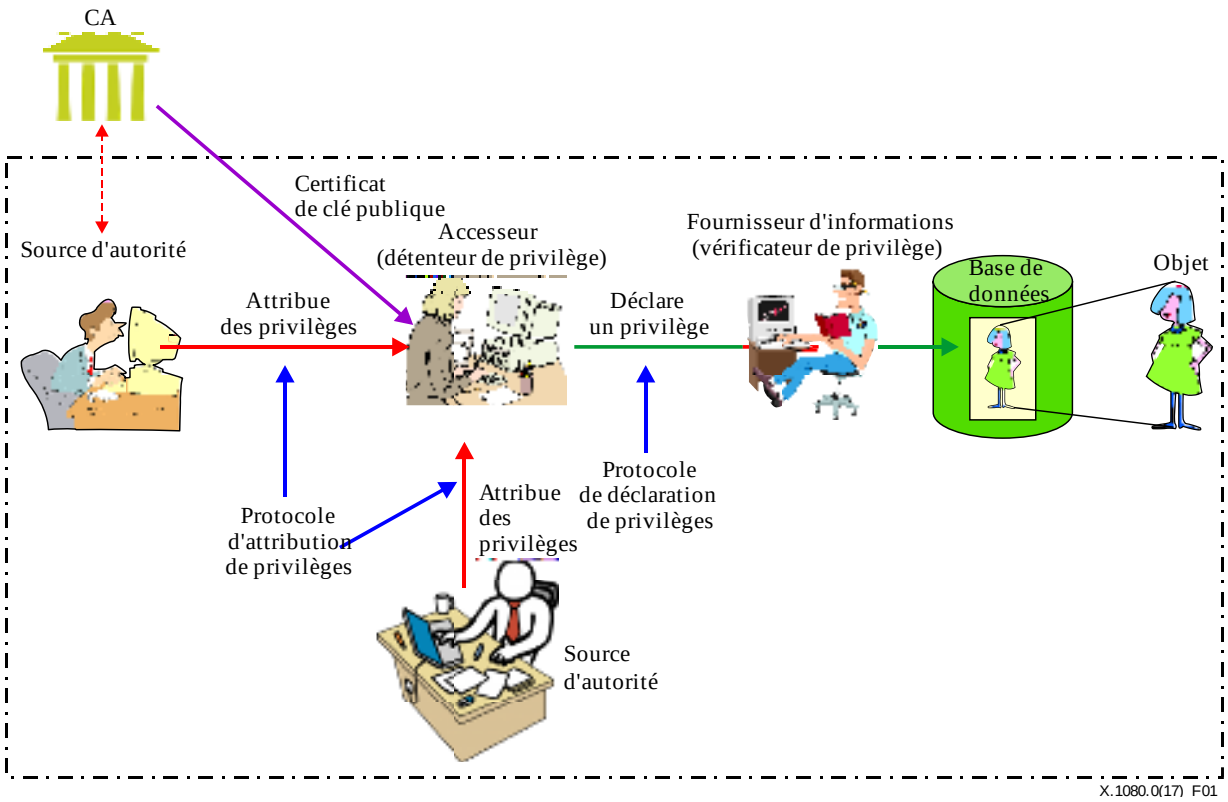

Figure 1 – Modèle de domaine unique de protection des données

La Figure 1 illustre les divers partenaires et protocoles d'un environnement de contrôle d'accès au sein d'un domaine unique de protection de données. Un domaine de protection de données comprend toutes les entités concernées par la protection qui relèvent d'une gestion commune.

Une entité, appelée accesseur, peut avoir besoin, dans le cadre de ses fonctions, d'être autorisée à effectuer des opérations sur des données concernant des objets détenus par un fournisseur d'informations. Par exemple, un médecin aura besoin d'accéder au dossier médical de son patient pour prescrire le meilleur traitement, tandis que les médecins qui ne s'occupent pas de ce patient n'ont aucunement besoin de ces informations.

L'autorisation d'effectuer une opération donnée sur des informations ne doit être accordée que si l'accesseur a le *droit* et le *réel besoin* d'effectuer cette opération, c'est-à-dire si le *privilège* nécessaire lui a été attribué.

La spécification des modalités d'attribution des privilèges aux entités n'entre pas dans le cadre de la présente Recommandation, qui se limite à décrire les outils nécessaires pour gérer les privilèges de façon sécurisée.

Dans un domaine de protection des données, il est nécessaire de définir une ou plusieurs autorités chargées d'attribuer les privilèges aux entités. L'expression "source d'autorité" (SOA) définie dans la Recommandation [UIT-T X.509] est utilisée ici pour désigner l'autorité ultime d'attribution des privilèges pour une zone particulière d'un domaine de protection des données.

Dans la présente Recommandation, les privilèges sont attribués aux accesseurs au moyen de certificats. Les privilèges peuvent être décrits dans des certificats d'attribut du composant **`attributes`** ou dans des certificats de clé publique de l'extension **`subjectDirectoryAttributes`**. En règle générale,

les privilèges permanents figurent dans des certificats de clé publique et les privilèges temporaires dans des certificats d'attribut.

La Figure 1 illustre la relation entre les différentes composantes d'un domaine unique de protection de données. Y figurent deux SOA, chacun étant chargé d'attribuer des privilèges à des détenteurs de privilège pour des aspects différents.

Certains privilèges peuvent être nécessaires pour les opérations quotidiennes de l'accesseur; ils ont alors un caractère plus permanent. D'autres en revanche, attribués pour gérer des situations particulières, sont de nature plus temporaire.

Lorsqu'un accesseur se voit attribuer des privilèges, il devient un détenteur de privilèges et il peut utiliser ses privilèges pour accéder à des informations via le protocole de déclaration de privilèges. Le vérificateur de privilège qui représente le fournisseur d'informations vérifie les privilèges déclarés avant d'autoriser l'accès aux données.

Une SOA peut transmettre des privilèges en utilisant des moyens locaux ou en les intégrant dans un certificat d'attribut signé qui sera transmis par le protocole d'attribution des privilèges, comme on peut le voir à la Figure 1.

6.2 Protection de données interdomaines

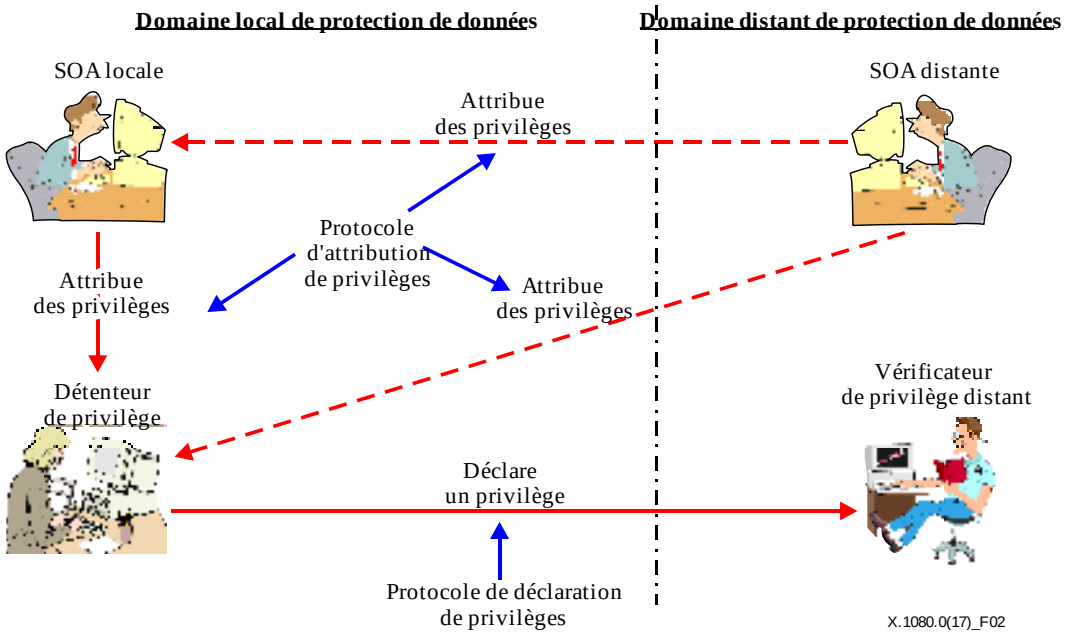

Figure 2 – Modèle de protection de données interdomaines

La Figure 2 illustre le cas où un accesseur a besoin d'accéder à des informations situées dans un autre domaine de protection de données. Par exemple, un médecin pourra souhaiter consulter des informations médicales essentielles sur son patient qui se trouvent dans un autre domaine.

Dans cet environnement, l'accesseur demande, directement ou par l'intermédiaire d'une SOA locale, à accéder à des informations particulières concernant un objet donné. La méthode utilisée pour transmettre la demande n'entre pas dans le cadre de la présente Recommandation. L'envoi d'un e-mail ou un appel téléphonique sont deux moyens possibles.

La SOA distante crée un certificat d'attribut, avec un attribut de type `accessService` qui contient le privilège nécessaire. Ce certificat est signé par la SOA distante. Le `holder` du certificat d'attribut peut être soit la SOA locale, soit l'entité qui a besoin du privilège.

Si le privilège est attribué à la SOA locale, cette SOA:

a) déchiffre le contenu si nécessaire;

b) vérifie la signature du contenu;

c) extrait le certificat d'attribut du contenu;

d) valide le certificat d'attribut ainsi extrait;

e) crée un nouveau certificat d'attribut avec les informations de privilège tirées du certificat d'attribut reçu et avec pour détenteur l'entité qui a besoin du privilège;

f) ce certificat d'attribut est ensuite transmis, en même temps que celui reçu de la SOA distante, à l'entité qui a besoin du privilège (c'est-à-dire, le détenteur du privilège).

6.3 Modèle orienté vers les services

Le contrôle d'accès appliqué à la télébiométrie est orienté vers les services en cela qu'un accesseur peut invoquer un ensemble de fonctions distinctes appelées *services d'accès*. Un service d'accès est une fonction fournie par un prestataire de services capable d'exécuter le service d'accès concerné. La capacité à manipuler des données de patient au sein d'un hôpital est un exemple de service d'accès.

Un identificateur d'objet est utilisé pour identifier un type de service d'accès particulier. La présente Recommandation ne définit pas de services d'accès, mais fournit les outils pour en créer. Les spécifications indiquées en référence peuvent définir des services d'accès correspondant à leur champ d'application.

Même si un accesseur a accès à un service particulier, il peut ne pas disposer du privilège couvrant tous les aspects de ce service.

6.4 Le modèle objet/attribut

Le modèle d'information défini dans la Recommandation [UIT-T X.501] est conçu pour gérer différentes structures de données. La présente Recommandation utilise ce modèle dans le cas particulier du contrôle d'accès.

Dans le modèle UIT-T X.501, les objets à protéger sont organisés en classes d'objets, de sorte que les objets appartenant à une classe possèdent des caractéristiques communes qui sont pertinentes dans un contexte particulier. Une classe d'objets est identifiée par un identificateur d'objet. La Recommandation [UIT-T 1080.1] élargit ce concept en attribuant des identificateurs d'objet à un groupe de classes d'objets pour des catégories en lien avec la télébiométrie.

La Recommandation [UIT-T X.521] définit aussi un ensemble de classes d'objets d'usage général. Dans les cas où une classe d'objets non directement liée à la télébiométrie est nécessaire, il convient, dans la mesure du possible, de recourir à une classe d'objets déjà définie.

Chaque objet est identifié au moyen d'un nom distinctif, tel que défini dans la Recommandation [UIT-T X.501]. Un nom distinctif consiste en un ou plusieurs composants de nom représentant la position de l'objet dans une hiérarchie d'objets.

L'information associée à un objet est modélisée par un ensemble d'attributs. Les attributs dotés de caractéristiques communes constituent un type d'attributs. Un type d'attribut est identifié par un identificateur d'objet. Un attribut est une séquence composée d'un identificateur d'objet identifiant le type d'attribut et d'une ou plusieurs valeurs de ce type. Un objet ne peut avoir qu'un attribut d'un type donné. La Recommandation [UIT-T X.520] définit un ensemble de types d'attribut d'usage général. Dans la mesure du possible, il convient d'utiliser des types d'attribut déjà définis. Selon les besoins, une spécification peut définir des types d'attribut supplémentaires.

Il convient, à chaque utilisation de la présente Recommandation, de fournir un schéma de correspondance entre ce modèle d'information et la structure effective de la base de données concernée. Cette correspondance s'établit aisément si l'information est conservée dans un annuaire

LDAP (protocole simple d'accès à l'annuaire) ou dans un annuaire sécurisé respectant les spécifications des Recommandations de la [série UIT-T X.500].

6.5 Principes élémentaires de contrôle d'accès

Ce paragraphe a pour objet de donner un aperçu des principes généraux qui régissent la mise en place d'un contrôle d'accès en télébiométrie. En règle générale, le contrôle d'accès concerne le droit ou le refus d'accéder à des données, sur la base de paramètres permanents. La présente Recommandation élargit cette démarche en tenant compte, en plus, des aspects relatifs au besoin de savoir. Pour ce faire, elle adopte une solution orientée vers les services, comme indiqué au paragraphe 6.3. Pour effectuer une opération donnée, un accesseur doit disposer d'un privilège qui l'autorise à accéder au service d'accès pertinent et d'un privilège qui l'autorise à accéder à l'information souhaitée.

Ce privilège donne accès à des objets appartenant à une ou plusieurs classes, avec une éventuelle limitation à certains objets nommés. Le type d'opération pouvant être effectué peut varier en fonction des classes d'objets et des objets nommés.

Les opérations sur les objets peuvent porter sur les attributs et sur les valeurs d'attribut. Les opérations autorisées peuvent varier selon les types des attributs.

6.6 Relation avec d'autres schémas de contrôle d'accès

Plusieurs types de contrôle d'accès ont été définis pour couvrir les différents besoins. L'objectif ici est de décrire brièvement divers schémas de contrôle d'accès et d'indiquer comment ils se positionnent par rapport à l'ACT.

6.6.1 Contrôle d'accès de base tel que défini dans la Recommandation UIT-T X.501

Le contrôle d'accès de base défini dans la Recommandation [UIT-T X.501] est utilisé pour protéger des informations figurant dans un annuaire tel que défini dans les Recommandations de la [série UIT-T X.500]. Il prévoit des listes de contrôle d'accès perfectionné qui indiquent en détail quels utilisateurs sont autorisés et comment ils le sont. Il diffère de la présente Recommandation en cela qu'il prend en charge le droit de savoir, mais pas le besoin de savoir.

6.6.2 Contrôle d'accès fondé sur des règles

Chacune des Recommandations [UIT-T X.501] et [b-UIT-T X.841] définit un type de contrôle d'accès fondé sur des règles. Dans ce type de contrôle d'accès, les données à protéger sont étiquetées avec des informations précisant quelle protection est requise, tandis que les utilisateurs qui accèdent aux données disposent d'informations certifiées associées, lesquelles sont intégrées à une demande d'accès, qui précise le niveau d'habilitation dont ils disposent pour accéder à certaines informations. Ce concept diffère de la présente Recommandation en cela qu'il nécessite l'étiquetage d'éléments de données mémorisés et qu'il prend en charge le droit de savoir, mais pas le besoin de savoir.

6.6.3 Contrôle d'accès fondé sur les rôles tel que défini pour les réseaux électriques intelligents

Le contrôle d'accès fondé sur les rôles tel que spécifié dans [b-IEC 62351-8] est axé sur les utilisateurs et sur les emplois qu'ils occupent. Un "rôle" est un ensemble de droits d'accès à des objets (actions pouvant être accomplies à certaines échéances). Un utilisateur peut avoir un ou plusieurs rôles. Dans le cas du contrôle d'accès, un rôle est une sorte d'intermédiaire qui permet de réduire le volume d'information nécessaire dans la liste ACL (liste de contrôle d'accès) en diminuant la granularité des informations relatives au contrôle d'accès. Les autorisations d'accès aux objets du système ne sont pas énumérées pour chaque utilisateur séparément, mais ces derniers se voient attribuer des rôles et les droits associés à chaque rôle ne sont décrits qu'une fois.

6.6.4 Contrôle d'accès fondé sur des attributs

Le contrôle d'accès fondé sur des attributs (ABAC) est un modèle de contrôle d'accès logique qui se démarque des autres, car l'accès aux objets y est contrôlé en évaluant des règles par rapport aux attributs des entités (sujet et objet), aux opérations et à l'environnement correspondant à une demande. Les systèmes ABAC peuvent mettre en œuvre le contrôle d'accès discrétionnaire aussi bien que le contrôle d'accès obligatoire. Le modèle ABAC permet d'effectuer un contrôle d'accès précis et de prendre en compte un grand nombre d'entrées discrètes dans les décisions de contrôle d'accès; il offre ainsi un plus grand ensemble de combinaisons possibles de ces variables, qui se déclinent en un ensemble plus étendu et plus abouti de règles à partir desquelles les politiques sont définies.

Le document [b-NIST 800-162] constitue une bonne introduction au modèle ABAC.

6.7 Protocoles: aperçu général

6.7.1 Protocole d'évaluation de privilèges

Le protocole d'évaluation de privilèges comprend un ensemble de types de contenu dans la syntaxe de message cryptographique (CMS). Chaque type d'accès nécessite un type de contenu de demande et un type de contenu de résultat.

Une instance de type de contenu est transmise au moyen de la syntaxe CMS avec le profil défini à l'Annexe B. Un module ASN.1 formel est fourni à l'Annexe C.

6.7.2 Protocole d'attribution de privilèges

Le protocole d'attribution de privilèges est utilisé pour transmettre des certificats d'attribut entre plusieurs SOA ou d'une SOA au détenteur de privilège.

Pour ce protocole, un couple unique de types de contenu est défini: le premier sert à transmettre le privilège sous la forme de certificats d'attribut, le second sert à renvoyer le résultat.

Une instance de type de contenu est transmise au moyen de la syntaxe CMS avec le profil défini à l'Annexe B. Un module ASN.1 formel est fourni à l'Annexe C.

6.8 Utilisation de CMS

Un profil CMS pour la télébiométrie figure à l'Annexe B.

Pour que la source d'information soit correctement authentifiée, les contenus des types définis dans la présente Recommandation doivent être encapsulés dans une instance du type de contenu **`signedData`**. Etant donné que des informations sensibles peuvent être transmises, il est, de plus, recommandé de les encapsuler dans une instance du type de contenu **`envelopedData`**. Le type de contenu **`ct-autEnvelopedData`** n'est pas utilisé dans la présente Recommandation.

6.9 Considérations touchant au certificat de clé publique

Un certificat d'attribut doit être signé par l'émetteur, qui utilise pour cela sa clé privée. Le certificat doit être vérifié au moyen du certificat de clé publique correspondant qui est fourni à l'émetteur.

En principe, la même clé privée pourrait être utilisée pour signer le message CMS au moyen du type de contenu **`signedData`**, ce qui faciliterait la vérification. Cela étant, l'utilisation de la même clé privée à différentes fins pose des problèmes de sécurité. Le recours à des clés privées différentes pour ces deux usages devrait être envisagé, bien que la présente Recommandation ne l'impose pas.

7 Communication des informations de privilège

7.1 Utilisation de certificats d'attribut

La Recommandation [UIT-T X.509] permet tant aux certificats de clé publique qu'aux certificats d'attribut d'acheminer des informations de privilège. Dans les deux cas, les spécifications de privilège sont stockées dans des attributs, tels qu'ils sont définis dans la Recommandation [UIT-T X.501]. Les types d'attribut utilisés à cette fin ne sont pas identiques à ceux utilisés pour modéliser les informations sur les objets. En effet, si les types d'attribut définis pour acheminer les informations sur les objets doivent rester aussi simples que possible, ceux destinés à transmettre les informations de privilège sont, par nature, assez complexes.

Lorsque les informations de privilège sont acheminées via un certificat d'attribut:

a) le composant **holder** porte l'identité de l'entité à laquelle des privilèges doivent être attribués. Lorsqu'un détenteur de privilège présente ce certificat d'attribut à un vérificateur, ce dernier authentifie l'accesseur pour vérifier qu'il s'agit bien du détenteur de privilège du certificat d'attribut;

b) le composant **issuer** contient le nom de la SOA ou le nom d'une autorité d'attribut (AA) qui a été mandatée pour attribuer des privilèges. Le vérificateur de privilège doit aussi obtenir et valider le certificat de clé publique de l'émetteur pour vérifier la signature du certificat d'attribut;

c) le composant **attributes** contient un attribut du type **accessService**, tel que défini au § 7.3.

Le certificat d'attribut doit être signé par la SOA qui a accordé le privilège ou par l'AA à laquelle la délivrance de certificat a été déléguée.

La validation sera plus simple si le détenteur et l'émetteur disposent de certificats de clé publique délivrés par la même autorité de certification (CA).

7.2 Utilisation de certificats de clé publique

Lorsque les informations de privilège sont acheminées via un certificat de clé publique:

a) le composant **subject** porte l'identité de l'accesseur auquel des privilèges ont été attribués. Lorsqu'un accesseur présente ce certificat de clé publique à un vérificateur, ce dernier l'authentifie;

b) le composant **issuer** contient le nom distinctif de la CA qui est chargée de délivrer le certificat de clé publique;

c) l'extension **subjectDirectoryAttributes** contient une instance du type d'attribut **accessService** (voir le § 7.3).

7.3 Le type d'attribut "service d'accès"

7.3.1 Syntaxe de l'attribut "service d'accès"

Un attribut de type **accessService** est censé être intégré dans le composant **attributes** d'un certificat d'attribut ou dans l'extension **subjectDirectoryAttributes** d'un certificat de clé publique. Cet attribut est défini par la syntaxe suivante:

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

Un attribut de type **AccessService** fournit les informations de privilège qui permettent à un vérificateur de privilège de déterminer si une demande d'accès doit ou non être honorée. Cet attribut peut prendre plusieurs valeurs, ce qui permet d'inclure, dans le même attribut, plusieurs types de service d'accès ainsi que les autorisations associées.

Une entité ne peut pas utiliser un service qui ne figure pas dans cet attribut.

Le type d'attribut **accessService** est défini par la syntaxe suivante:

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

Les composants d'une valeur de type **AccessService** sont les suivants:

a) le composant **serviceId** identifie le type de service d'accès pour lequel l'accesseur détient un privilège;

b) le composant **objectDef** spécifie les classes d'objets pour lesquelles un privilège est attribué au détenteur du certificat d'attribut pour le type de service concerné. Il contient une séquence d'éléments, un pour chaque classe d'objets pour laquelle un privilège a été attribué. Pour le service concerné, le détenteur de privilège n'a pas accès aux classes d'objets qui ne figurent pas dans le composant **objectDef**.

7.3.2 Sélection d'objets

La sélection des objets vis-à-vis desquels l'accesseur dispose de privilèges est spécifiée par une instance du type de données **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,
      ... },
    ... },
  ... }
```

Une valeur du type **ObjectSel** indique le privilège associé à chaque classe d'objets pour laquelle un privilège a été attribué pour le type de service d'accès concerné. Cette valeur est constituée des composants suivants:

– le composant **objectClass** indique la classe d'objets pour laquelle des privilèges sont attribués;

– le composant **objSelect** indique quels objets de la classe sont concernés par l'attribution de privilèges. Il a deux formes possibles:

  a) la forme **allObj** doit être retenue si les privilèges attribués s'appliquent de la même manière à tous les objets de la classe. Elle contient une instance du type de données **TargetSelect**;

  b) la forme **objectNames** doit être retenue si les privilèges ne s'appliquent qu'à certains objets de la classe d'objets identifiée. Elle peut contenir plusieurs éléments, chacun comprenant les composants suivants:

    i) le composant **object** indique un ou plusieurs objets pour lesquels des privilèges ont été attribués. Il a deux formes possibles:

      • la forme **names** permet d'indiquer le nom d'un ou plusieurs objets auxquels les privilèges s'appliquent;

      • la forme **subtree** sera retenue pour indiquer un groupe d'objets dans lequel chaque objet possède un nom distinctif identique à celui de cette forme ou dans lequel les premiers composants de nom de chaque objet sont identiques à ceux de ce nom distinctif;

ii) le composant **select** contient une instance du type de données **TargetSelect**.

Le type de données **TargetSelect** est défini par la syntaxe suivante:

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

Le type de données **TargetSelect** comprend les composants optionnels suivants, l'un d'eux au moins devant être présent:

a) lorsqu'il est présent, le composant **objOper** indique les opérations qui sont autorisées sur les objets de la classe d'objets ou sur certains objets. S'il n'est pas présent, aucune opération n'est autorisée sur l'ensemble des objets;

b) lorsqu'il est présent, le composant **attrSel** contient une valeur du type de données **AttributeSel** (voir le § 7.3.3). S'il n'est pas présent, aucune opération sur les attributs des objets n'est autorisée.

7.3.3 Sélection des types d'attribut

```
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,
      ... },
    ... },
  ... }
```

Le type de données **AttributeSel** précise les types d'attribut pour lesquels le privilège s'applique. Il est constitué des composants suivants:

– Le composant **attSelect** a deux formes possibles:

  a) la forme **allAttr** doit être retenue si le privilège s'applique à tous les attributs de l'objet/des objets. Il est constitué d'un seul composant:

    i) le composant **attrOper1** précise les opérations qui peuvent être effectuées sur les attributs;

  b) la forme **attributes** doit être retenue si le privilège ne s'applique qu'à certains attributs de l'objet/des objets. L'accesseur ne dispose d'aucun privilège vis-à-vis des types d'attribut qui ne sont pas donnés dans la liste et il ne doit pas avoir connaissance de ces types d'attributs. Cette forme est constituée des composants suivants:

    i) le composant **select** précise un ou plusieurs types d'attribut pour lesquels les privilèges s'appliquent;

    ii) le composant **attrOper2** précise les opérations qui peuvent être effectuées sur les attributs.

7.4 Opérations sur des objets dans leur ensemble

Le type de données suivant est utilisé pour indiquer les opérations qui peuvent être effectuées sur un objet:

```
ObjectOperations ::= BIT STRING {
  read              (0),
```

```
  add              (1),
  modify           (2),
  delete           (3),
  rename           (4),
  discloseOnError (5) }
```

L'autorisation **read** doit être positionnée lorsque l'accesseur est autorisé à lire des informations appartenant à un objet.

L'autorisation **add** doit être positionnée lorsque l'accesseur est autorisé à ajouter de nouveaux objets de la classe d'objets concernée. Cette autorisation doit être positionnée pour tous les objets d'une classe donnée. Pour chaque attribut à ajouter à l'objet, la permission **add** doit être positionnée pour le type d'attribut correspondant (voir le § 7.5).

L'autorisation **modify** doit être positionnée lorsque l'accesseur est autorisé à modifier un objet existant. L'accesseur doit disposer de l'autorisation **modify** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés à modifier. Si l'accesseur ajoute des attributs, il doit disposer de l'autorisation **add** pour les types d'attribut concernés. S'il supprime des attributs, il doit disposer de l'autorisation **delete** pour les types d'attribut concernés. S'il modifie des attributs, il doit disposer de l'autorisation **modify** pour les types d'attribut concernés. S'il supprime des attributs, il doit disposer de l'autorisation **deleteValue** pour les types d'attribut concernés. S'il remplace des attributs, il doit disposer de l'autorisation **replaceAttribute** pour les types d'attribut concernés.

L'autorisation **delete** doit être positionnée lorsque l'accesseur est autorisé à supprimer un objet existant. L'accesseur doit disposer de l'autorisation **delete** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés à supprimer.

L'autorisation **rename** doit être positionnée lorsque l'accesseur est autorisé à renommer un ou des objets existants. L'accesseur doit disposer de l'autorisation **rename** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés à renommer.

L'élément **discloseOnError** doit être positionné pour autoriser que l'accesseur ait connaissance de l'existence de l'objet lorsqu'une opération échoue.

7.5 Opérations sur des attributs

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

L'autorisation **read** doit être positionnée pour chacun des types d'attributs souhaités pour que l'accesseur soit autorisé à lire ces attributs. L'accesseur doit disposer de l'autorisation **read** pour la classe d'objets concernée dans son ensemble ou pour les objets nommés qui sont concernés. De plus, il doit disposer de l'autorisation **read** pour tous les attributs de ces classes d'objets ou avoir accès au(x) type(s) d'attribut concerné(s).

L'autorisation **compare** doit être positionnée lorsque l'accesseur est autorisé à comparer un ou plusieurs attributs. Il doit disposer de l'autorisation **read** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés. De plus, il doit disposer de l'autorisation **compare** pour tous les attributs des classes d'objets autorisées ou pour le ou les types d'attribut concernés.

L'autorisation **add** doit être positionnée lorsque l'accesseur est autorisé à ajouter un ou plusieurs attributs. L'accesseur doit avoir l'autorisation **modify** pour la classe d'objets dans son ensemble ou

pour le ou les objets nommés. De plus, il doit disposer de l'autorisation **add** pour les types d'attribut concernés.

L'autorisation **modify** doit être positionnée lorsque l'accesseur est autorisé à modifier un attribut d'un type particulier. De plus, l'accesseur doit disposer de l'autorisation **modify** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés.

L'autorisation **delete** doit être positionnée lorsque l'accesseur est autorisé à supprimer un ou plusieurs attributs. De plus, l'accesseur doit disposer de l'autorisation **modify** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés.

L'autorisation **deleteValue** doit être positionnée lorsque l'accesseur est autorisé à supprimer une ou plusieurs valeurs d'un attribut du ou des types d'attribut. De plus, l'accesseur doit disposer de l'autorisation **modify** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés.

L'autorisation **replaceAttribute** doit être positionnée lorsque l'accesseur est autorisé à remplacer un attribut d'un type donné par un attribut de même type. De plus, l'accesseur doit disposer de l'autorisation **modify** pour la classe d'objets dans son ensemble ou pour le ou les objets nommés.

L'élément **discloseOnError** doit être positionné pour autoriser que l'accesseur ait connaissance de l'existence d'un attribut lorsqu'une opération échoue. De plus, il doit disposer de l'autorisation **discloseOnError** pour l'objet dans son ensemble.

7.6 Gestion des erreurs

Comme indiqué à l'Annexe A.5, des erreurs peuvent survenir à l'utilisation de la CMS. Lorsqu'une erreur est détectée, il est inutile de poursuivre le contrôle. Les erreurs CMS sont renvoyées dans le résultat de l'accès qui a été demandé.

Des erreurs peuvent aussi apparaître dans le résultat de la demande d'accès elle-même.

L'erreur est transmise par une instance du type de données **AccessdErr** défini au § 8.10.

8 Protocole de déclaration de privilèges

8.1 Présentation générale

L'ensemble d'objets informationnels suivant contient tous les types de contenu définis qui sont représentés par les objets informationnels définis par la présente Recommandation.

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

Les types de contenu spécifiés par l'ensemble **ActContentTypes** constituent le protocole de déclaration de privilèges, qui comprend un certain nombre d'opérations d'accès différentes, tel que défini aux § 8.4 à 8.9.

8.2 Composants de demande communs

Les composants suivants figurent dans toutes les demandes:

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

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

Les paramètres de demande communs sont les suivants:

a) le composant **attrCert**, lorsqu'il est présent, précise le certificat d'attribut ou le chemin de certification d'attribut dans lequel figure le privilège de l'accesseur. Si ce composant est absent, le privilège doit être indiqué dans le certificat de clé publique d'entité finale correspondant à l'accesseur;

b) le composant **serviceId** doit indiquer le type de service à invoquer;

c) le composant **invokId** doit prendre la valeur zéro pour la première opération invoquée, puis être incrémenté de un pour chaque opération invoquée suivante. L'intervalle de valeurs doit être suffisamment grand pour que, pendant une longue période, la même valeur ne soit pas réutilisée pour la communication entre deux entités. Ce composant est mis en place pour détecter les demandes manquantes ainsi que les attaques par répétition. Le destinataire d'une demande doit utiliser la même valeur dans sa réponse pour que l'accesseur puisse faire le lien entre un résultat et la demande correspondante.

8.3 Accès à un service

Tous les types d'opération définis aux § 8.4 à 8.9 nécessitent un accès à un service particulier. Le vérificateur de privilège (destinataire) doit contrôler que le privilège attribué à l'accesseur dans le certificat d'attribut ou le certificat de clé publique associé autorise l'accès au service demandé.

Si l'accesseur n'a pas l'autorisation d'invoquer le type de service demandé ou que le fournisseur de services ne prend pas en charge ce type de service, un code d'erreur **noSuchService** doit être renvoyé.

Si l'accesseur est autorisé à invoquer le service, il convient de contrôler si l'opération demandée est compatible ou non avec le type de service; si ce n'est pas le cas, un code d'erreur **invalidOperationForService** doit être renvoyé.

8.4 Opération de lecture

Une opération de lecture se compose d'une demande de lecture et du résultat de lecture correspondant.

Une demande de lecture est acheminée sous la forme d'une instance du type de contenu **readRequest** et le résultat de lecture sous la forme d'une instance du type de contenu **readResult**.

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

L'accesseur utilise le type de contenu **readRequest** pour lire des informations concernant un objet donné.

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

Le type de données **ReadRequest** spécifie la syntaxe du contenu effectif et contient les composants suivants:

a) le composant **object** contient le nom distinctif de l'objet à propos duquel des informations sont demandées;

b) le composant **selection** spécifie le type d'information que demande l'accesseur (voir le § 8.11).

La demande de lecture échoue si elle concerne un objet inconnu; dans ce cas, un code d'erreur **noSuchObject** est renvoyé.

La demande de lecture échoue si l'accesseur ne dispose pas de l'autorisation **read** pour l'objet concerné compte tenu du privilège qui lui est attribué. Si l'opération **read** n'est pas autorisée et que l'accesseur dispose de l'autorisation **discloseOnError** pour l'objet concerné, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noSuchObject** est renvoyé.

Pour chaque attribut qui doit être renvoyé, l'autorisation **read** doit être positionnée pour le type de cet attribut. Si l'accesseur ne dispose pas de l'autorisation **read** pour un type d'attribut donné, aucun attribut de ce type n'est renvoyé dans le résultat, auquel cas, la demande échoue. Si l'accesseur dispose de l'autorisation **discloseOnError** pour toutes les demandes d'attribut, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noInformation** est renvoyé.

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

Le vérificateur de privilège utilise une instance du type de contenu **readResult** pour renvoyer l'information demandée ou pour signaler une erreur.

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

Le type de données **ReadResult** spécifie la syntaxe du contenu et contient les composants suivants:

a) le composant **object** contient le nom de l'objet à propos duquel des informations ont été demandées;

b) le composant **result** contient le résultat de la demande de lecture. Il a deux formes possibles:

- La forme **success** est retenue si une information doit être renvoyée; dans ce cas, elle contient une instance du type de données **ObjectInformation** (voir le § 8.12). L'information renvoyée est l'intersection entre ce que l'accesseur a demandé et les informations qu'il est autorisé à lire;
- La forme **failure** est retenue dans le cas où une erreur doit être renvoyée.

8.5 Opération de comparaison

Une opération de comparaison se compose d'une demande de comparaison et du résultat de comparaison correspondant.

Une demande de comparaison est acheminée sous la forme d'une instance du type de contenu **compareRequest** et le résultat de comparaison sous la forme d'une instance du type de contenu **compareResult**.

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

Une instance du type de contenu **compareRequest** est utilisée pour comparer une valeur prétendue d'un type d'attribut donné avec une valeur d'attribut du même type appartenant à un objet particulier.

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

Le type de données **CompareRequest** précise le contenu effectif et contient les composants suivant, en plus de ceux définis au § 8.2:

a) le composant **object** contient le nom distinctif de l'objet dont une valeur d'attribut doit être comparée;

b) le composant **purported** contient une combinaison d'un type d'attribut et d'une valeur d'attribut à comparer avec un attribut du même type contenu dans l'objet en question.

La demande de comparaison échoue si elle concerne un objet inconnu; dans ce cas, un code d'erreur **noSuchObject** est renvoyé.

La demande de comparaison échoue si l'accesseur ne dispose pas de l'autorisation **read** pour l'objet concerné compte tenu du privilège qui lui est attribué. Si l'autorisation **read** n'a pas été attribuée à l'objet et que l'accesseur dispose de l'autorisation **discloseOnError** pour cet objet, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noSuchObject** est renvoyé.

La demande de comparaison échoue si l'accesseur ne dispose pas de l'autorisation **compare** pour le type d'attribut concerné. Si l'accesseur dispose de l'autorisation **discloseOnError** pour le type d'attribut, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noInformation** est renvoyé.

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

Le vérificateur de privilège utilise une instance du type de contenu **compareResult** pour renvoyer l'information demandée ou pour signaler une erreur.

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

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

Le type de données **CompareResult** spécifie la syntaxe du contenu effectif et contient les composants suivants:

a) le composant **object** contient le nom distinctif de l'objet à propos duquel une demande de comparaison devait être effectuée;

b) le composant **result** contient le résultat de la demande d'accès. Il a deux formes possibles:

   – si la forme **success** est retenue, une instance du type de données **CompareOK** est renvoyée avec les composants suivants:

      i) le composant **matched** prend la valeur **TRUE** si un attribut du type d'attribut ou l'un de ses sous-types contient une valeur égale à celle figurant dans la demande. De plus,

le composant **matchedSubtype** doit être présent et avoir pour valeur **TRUE** dans le cas d'une correspondance de sous-types. Le composant **matched** prend la valeur **FALSE** si aucune correspondance n'a été trouvée ni pour le type d'attribut ni pour l'un de ses sous-types;

– la forme **failure** est retenue dans le cas où une erreur doit être renvoyée.

8.6 Opération d'ajout

Une opération d'ajout se compose d'une demande d'ajout et du résultat d'ajout correspondant.

Une demande d'ajout est acheminée sous la forme d'une instance du type de contenu **addRequest** et le résultat d'ajout sous la forme d'une instance du type de contenu **addResult**.

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

Une instance du type de contenu **addRequest** est utilisée pour ajouter un nouvel objet au système d'information.

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

Le type de données **AddRequest** spécifie la syntaxe du contenu et contient les composants suivants:

a) le composant **object** contient le nom distinctif du nouvel objet à ajouter;

b) le composant **attr**, lorsqu'il est présent, contient un ou plusieurs attributs à associer au nouvel objet.

Si l'accesseur ne dispose pas de l'autorisation **add** pour la classe d'objets, un code d'erreur **insufficientAccessRight** est renvoyé.

Si un objet possédant le nom distinctif fourni existe déjà et que l'accesseur dispose de l'autorisation **discloseOnError**, un code d'erreur **objectAlreadyExists** est renvoyé. Sinon, un code d'erreur **insufficientAccessRight** est renvoyé.

Si l'accesseur ne dispose pas de l'autorisation **add** pour tous les attributs à intégrer avec l'objet, la demande échoue. Si l'accesseur dispose de l'autorisation **discloseOnError** pour tous les types d'attribut donnés dans la liste, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noInformation** est renvoyé.

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

Le vérificateur de privilège utilise une instance du type de contenu **addResult** pour renvoyer l'information demandée ou pour signaler une erreur.

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

Le type de données **AddResult** spécifie la syntaxe du contenu et contient les composants suivants:

a) la forme **success** est retenue si l'objet a été ajouté;

b) la forme **failure** est retenue dans le cas où une erreur doit être renvoyée.

8.7 Opération de suppression

Une opération de suppression se compose d'une demande de suppression et du résultat de suppression correspondant.

Une demande de suppression est acheminée sous la forme d'une instance du type de contenu **deleteRequest** et le résultat de suppression sous la forme d'une instance du type de contenu **deleteResult**.

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

Une instance du type de contenu **deleteRequest** est utilisée pour supprimer un objet existant du système d'information.

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

Le type de données **DeleteRequest** spécifie la syntaxe du contenu effectif; il est composé d'un seul élément:

a) le composant **object** contient le nom distinctif de l'entrée à supprimer.

Si l'objet à supprimer n'existe pas, un code d'erreur **noSuchObject** est renvoyé.

Si l'accesseur ne dispose pas de l'autorisation **delete** pour la classe d'objets et qu'il dispose de l'autorisation **discloseOnError** pour l'objet, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noSuchObject** est renvoyé.

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

Le vérificateur de privilège utilise une instance du type de contenu **deleteResult** pour renvoyer l'information demandée ou pour signaler une erreur.

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

Une instance du type de données **DeleteResult** peut prendre deux formes:

a) la forme **success** est retenue si l'objet a effectivement été supprimé;

b) la forme **failure** est retenue dans le cas où une erreur doit être renvoyée.

8.8 Opération de modification

Une opération de modification se compose d'une demande de modification et du résultat de modification correspondant.

Une demande de modification est acheminée sous la forme d'une instance du type de contenu **modifyRequest** et le résultat de modification sous la forme d'une instance du type de contenu **modifyResult**.

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

Une instance du type de contenu **modifyRequest** est utilisée pour modifier un objet existant.

```
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}},
  ... }
```

Le type de données **ModifyRequest** spécifie la syntaxe du contenu effectif et contient les composants suivants:

a) Le composant **object** contient le nom distinctif de l'objet à modifier:

- si l'objet n'existe pas, un code d'erreur **noSuchObject** est renvoyé;
- si l'accesseur ne dispose pas de l'autorisation **modify** pour l'objet et qu'il dispose de l'autorisation **discloseOnError** pour l'objet, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noSuchObject** est renvoyé.

b) Le composant **changes** contient les informations nécessaires à la modification d'un ou de plusieurs attributs:

- La forme **addAttribute** contient un nouvel attribut à ajouter:
  - i) si l'accesseur ne dispose pas de l'autorisation **add** pour le type d'attribut, un code d'erreur **insufficientAccessRight** est renvoyé;
  - ii) si un attribut du type concerné existe déjà et que l'accesseur dispose de l'autorisation **discloseOnError** pour le type d'attribut en question, un code d'erreur **attributeAlreadyExists** est renvoyé. Sinon, un code d'erreur **insufficientAccessRight** est renvoyé.
- La forme **deleteAttribute** identifie l'attribut à supprimer:
  - i) si l'accesseur ne dispose pas de l'autorisation **delete** pour le type d'attribut, un code d'erreur **insufficientAccessRight** est renvoyé;
  - ii) s'il n'existe pas d'attribut du type concerné, un code d'erreur **noSuchAttribute** est renvoyé;
- La forme **addValues** identifie un attribut existant par le type d'attribut. Les valeurs à ajouter à l'attribut sont celles qui figurent dans cette forme:
  - i) si l'objet ne contient aucun attribut du type donné et que l'accesseur dispose de l'autorisation **discloseOnError**, un code d'erreur **noSuchAttribute** est renvoyé. Sinon, un code d'erreur **insufficientAccessRight** est renvoyé;
  - ii) si l'accesseur ne dispose pas de l'autorisation **addValue**, un code d'erreur **insufficientAccessRight** est renvoyé;
  - iii) si l'accesseur tente d'ajouter une valeur existante et qu'il dispose de l'autorisation **discloseOnError**, un code d'erreur **attributeAlreadyExists** est renvoyé. Sinon, un code d'erreur **insufficientAccessRight** est renvoyé.
- La forme **deleteValues** identifie un attribut par le type d'attribut. Les valeurs à supprimer à l'attribut sont celles qui figurent dans cette forme:
  - i) si l'objet ne contient aucun attribut du type donné et que l'accesseur dispose de l'autorisation **discloseOnError**, un code d'erreur **noSuchAttribute** est renvoyé. Sinon, un code d'erreur **insufficientAccessRight** est renvoyé;

ii) si l'accesseur ne dispose pas de l'autorisation **deleteValue** et qu'il dispose de l'autorisation **discloseOnError** pour l'objet, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noSuchAttributeValue** est renvoyé;

iii) si l'accesseur tente de supprimer une valeur d'attribut qui n'existe pas, un code d'erreur **noSuchAttributeValue** est renvoyé.

– La forme **replaceAttribute** remplace un attribut existant par un nouvel attribut du même type:

i) si l'objet ne contient aucun attribut du type donné et que l'accesseur dispose de l'autorisation **discloseOnError**, un code d'erreur **noSuchAttribute** est renvoyé. Sinon, un code d'erreur **insufficientAccessRight** est renvoyé;

ii) si l'accesseur ne dispose pas de l'autorisation **replaceAttribute** et qu'il dispose de l'autorisation **discloseOnError** pour l'objet, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noSuchAttribute** est renvoyé.

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

Le vérificateur de privilège utilise une instance du type de contenu **modifyResult** pour renvoyer l'information demandée ou pour signaler une erreur.

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

Une instance du type de données **ModifyResult** peut prendre deux formes:

a) La forme **success** est retenue si l'objet a effectivement été modifié.

b) La forme **failure** est retenue dans le cas où une erreur doit être renvoyée.

8.9 Opération de renommage d'un objet

Une opération de renommage d'un objet se compose d'une demande de renommage et du résultat de renommage correspondant.

Une demande de renommage est acheminée sous la forme d'une instance du type de contenu **renameRequest** et le résultat de renommage sous la forme d'une instance du type de contenu **renameResult**.

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

Une instance du type de contenu **renameRequest** est utilisée pour modifier le nom d'un objet existant.

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

Le type de données **RenameRequest** spécifie la syntaxe du contenu effectif et contient les composants suivants:

a) Le composant **object** précise le nom distinctif actuel de l'objet à renommer.

b) Le composant **new** contient le nouveau nom distinctif de l'objet.

Si l'objet à renommer n'existe pas, un code d'erreur **noSuchObject** est renvoyé.

Si l'accesseur ne dispose pas de l'autorisation **rename** pour l'objet nommé et qu'il dispose de l'autorisation **discloseOnError** pour l'objet, un code d'erreur **insufficientAccessRight** est renvoyé. Sinon, un code d'erreur **noSuchObject** est renvoyé.

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

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

Une instance du type de données **RenameResult** peut prendre deux formes:

a) La forme **success** est retenue si l'objet a effectivement été modifié.

b) La forme **failure** est retenue dans le cas où une erreur doit être renvoyée.

8.10 Gestion des erreurs

Lorsqu'une condition d'exception survient pendant le traitement d'une demande, un code d'erreur doit être renvoyé par le destinataire, qui insère pour cela une instance du type de données **AccessErr** dans le résultat.

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

La forme **cmsErr** est retenue si une condition d'exception est survenue pendant l'évaluation des types de contenu CMS définis (voir le § A.5).

La forme **pbactErr** est retenue si aucune erreur n'a été détectée lors de l'évaluation des instances des types de contenu CMS, mais qu'une erreur a été détectée dans une instance encapsulée d'un type de contenu PBACT spécifique.

8.11 Sélection des informations

Le type de données **InformationSelection** est utilisé pour indiquer quelles informations sont demandées lors d'une demande de lecture ou de modification.

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

Le type de données **InformationSelection** contient les composants suivants:

a) Le composant **attributes** indique quels attributs doivent être renvoyés. Il a deux formes possibles:

– la forme **allAttributes** est retenue si la demande de l'accesseur porte sur l'ensemble des informations concernant l'objet; ou

– la forme **select** est retenue lorsque la demande porte sur un sous-ensemble des attributs.

b) Le composant **infoTypes** possède les éléments d'énumération suivants:

– l'élément d'énumération **attributeTypesOnly** est retenu s'il ne faut renvoyer que des types d'attribut. L'accesseur doit alors disposer, dans ses privilèges, de l'autorisation **read** pour le type d'attribut concerné. Dans le cas contraire, le type d'attribut en question est retiré du résultat. Si, pour cette raison, aucune information ne peut être renvoyée, la demande échoue;

– l'élément d'énumération **attributeTypesAndValues** est retenu si le type et les valeurs doivent être renvoyés pour le privilège en vigueur. L'accesseur doit alors disposer, dans ses privilèges, de l'autorisation **read** pour le type d'attribut concerné. Dans le cas contraire, le type d'attribut et la valeur en question sont retirés du résultat. Si, pour cette raison, aucune information ne peut être renvoyée, la demande échoue.

8.12 Information concernant un objet

Lorsqu'une information concernant un objet doit être renvoyée, elle doit l'être sous la forme d'une instance du type de données suivant:

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

Le composant **object** contient le nom distinctif de l'objet pour lequel des informations sont renvoyées.

Le composant **info** contient un ensemble d'attributs dans lesquels figurent les informations ou l'ensemble des types d'attribut demandés.

S'il n'y a aucune information à renvoyer, la demande échoue.

8.13 Codes d'erreur définis

Les codes d'erreur applicables aux types de contenu PBACT spécifiques sont définis ci-dessous.

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

a) Le code d'erreur **noSuchService** est renvoyé si l'accesseur indique un service pour lequel il ne dispose pas d'autorisation et qu'il n'est pas autorisé à connaître, ou un service qui n'est pas pris en charge.

b) Le code d'erreur **invalidOperationForService** est renvoyé si une opération demandée ne s'applique pas au service demandé.

c) Le code d'erreur **insufficientAccessRight** est renvoyé lorsque l'accesseur demande un service pour lequel il n'a pas d'autorisation ou qu'il souhaite effectuer une opération qui ne lui est pas autorisée compte tenu du service auquel il faut accéder.

d) Le code d'erreur **noSuchObject** est renvoyé si un accesseur essaie d'accéder à un objet qui n'existe pas ou à un objet dont il n'est pas autorisé à connaître l'existence.

e) Le code d'erreur **noSuchAttribute** est renvoyé si un accesseur essaie d'accéder à un attribut qui n'existe pas ou à un attribut dont il n'est pas autorisé à connaître l'existence.

f) Le code d'erreur **noSuchAttributeValue** est renvoyé si un accesseur essaie d'accéder à une valeur d'attribut qui n'existe pas ou à une valeur d'attribut dont il n'est pas autorisé à connaître l'existence.

g) Le code d'erreur **objectAlreadyExists** est renvoyé lorsque l'on tente d'ajouter un objet dont le nom distinctif est le même que celui d'un objet existant, à condition que l'accesseur soit autorisé à connaître l'existence de cet objet.

h) Le code d'erreur **attributeValueAlreadyExists** est renvoyé lorsque l'on tente d'ajouter un attribut d'un type qui existe déjà dans l'objet, à condition que l'accesseur soit autorisé à connaître l'existence de ce type d'attribut dans l'objet.

i) Le code d'erreur **attributeValueAlreadyExists** est renvoyé lorsque l'on tente d'ajouter un attribut à un objet qui contient déjà un attribut de même type, à condition que l'accesseur soit autorisé à connaître l'existence de cet attribut.

j) Le code d'erreur **noInformation** est renvoyé si une information est demandée, mais qu'elle n'est pas disponible ou que l'accesseur n'est pas autorisé à en connaître l'existence.

9 Protocole d'attribution de privilèges

9.1 Champ d'application du protocole

Le protocole d'attribution de privilèges est utilisé pour attribuer des privilèges:

a) provenant d'une SOA, à une AA intermédiaire en vue d'une délégation ultérieure;

b) provenant d'une SOA, directement à une entité qui utilise les privilèges attribués pour déclarer ces privilèges;

c) provenant d'une AA, directement à une entité qui utilise les privilèges attribués pour déclarer ces privilèges; et

d) d'une AA à une autre AA, lorsqu'il y a plusieurs AA sur le chemin qui relie une SOA au détenteur de privilège.

NOTE – Il est recommandé de faire en sorte que le chemin de délégation soit le plus court possible.

9.2 Types de contenu

Le protocole d'attribution de privilèges utilise deux types de contenu: l'un pour attribuer des privilèges, l'autre pour confirmer l'attribution.

9.2.1 Type de contenu "demande d'attribution de privilège"

Une demande d'attribution de privilège est acheminée dans une instance du type de contenu **privAssignRequest**.

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

La syntaxe du contenu est définie par le type de données suivant:

```
PrivAssignRequest ::= SEQUENCE {
```

```
  attrCerts   AttributeCertificates OPTIONAL,
  ... }
```

Ce type de données est constitué d'un seul composant, qui contient une séquence de certificats d'attribut. Si la demande d'attribution de privilège est envoyée par une SOA, la séquence contient un seul certificat. S'il n'y a qu'une seule AA entre la SOA et le détenteur de privilège, la demande faite par l'AA à ce dernier contient deux certificats d'attribut, celui délivré par la SOA et celui délivré par l'AA. Un certificat d'attribut supplémentaire est nécessaire pour chaque AA additionnelle située entre la SOA et le détenteur de privilège.

9.2.2 Type de contenu "résultat d'attribution de privilège"

Un résultat d'attribution de privilège est acheminé dans une instance du type de contenu **privAssignResult**.

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

La syntaxe du contenu est définie par le type de données suivant:

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

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

9.2.3 Codes d'erreur définis

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

Annexe A

Attribution des identificateurs d'objet pour les Recommandations UIT-T de la série 1080

(Cette annexe fait partie intégrante de cette Recommandation.)

A.1 Niveau supérieur de l'arbre des identificateurs d'objet

L'annexe A à la Recommandation [UIT-T X.1081] attribue des arcs sous l'arc attribué à la télébiométrie, autrement dit:

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

Sous cet arc, la Recommandation [UIT-T 1081] attribue l'arc suivant pour la télésanté:

```
id-th OBJECT IDENTIFIER ::= { id-telebio th(3) }
```

La Recommandation [UIT-T X.1080.1] a attribué plusieurs arcs sous l'arc `id-th`. L'arc '0' est attribué pour les modules ASN.1 définis dans la Recommandation [UIT-T 1080.1]. D'autres arcs sont attribués pour les catégories d'entité. La présente Recommandation spécifie que l'arc '0' est aussi utilisé pour l'attribution d'identificateurs d'objet dans le contexte des Recommandations UIT-T de la série X.1080 en général. Pour éviter les collisions, la valeur '10' est utilisée pour l'attribution d'identificateurs d'objet dans le contexte des Recommandations UIT-T de la série X.1080.

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

Les arcs suivants sont attribués pour les différentes parties des Recommandations UIT-T de la série X.1080:

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

La répartition de ces arcs se présente comme suit:

– les modules sont associés à l'arc '0';

– les types de contenu CMS sont associés à l'arc '1';

– les types d'attribut sont associés à l'arc '2'.

Une partie peut attribuer des arcs supplémentaires en fonction de ses besoins.

Dans le cadre de la présente Recommandation, l'arc suivant est attribué pour les modules:

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

L'arc suivant est attribué pour les types de contenu CMS:

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

L'arc suivant est attribué pour les types d'attribut utilisés pour attribuer des privilèges:

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

A.2 Identificateurs d'objet pour les types de contenu CMS

Les identificateurs d'objet suivants sont attribués pour les types de contenu définis pour le protocole d'attribution de privilèges et pour le protocole d'évaluation de privilèges:

```
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 Identificateurs d'objet pour les types d'attribut de privilège

```
id-at-accessSer   OBJECT IDENTIFIER ::= { id-pbactPrivAttr 1 }
```

Annexe B

Profil de syntaxe de message cryptographique

(Cette annexe fait partie intégrante de cette Recommandation.)

B.1 Généralités

La syntaxe de message cryptographique (CMS), qui est définie dans [IETF RFC 5652], spécifie les capacités de communication qui permettent d'assurer l'intégrité, l'authentification et la confidentialité des données. Le document [IETF RFC 5083] contient des spécifications supplémentaires. Les documents [IETF RFC 5911] et [IETF RFC 6268] définissent de nouveaux modules ASN.1 pour CMS. La présente Annexe définit un profil CMS utilisé par les spécifications de la télébiométrie en référence à ces spécifications.

CMS est une spécification polyvalente qui peut être utilisée dans de nombreux environnements. Le présent profil n'utilise pas toutes les capacités de la syntaxe CMS. Il comprend les types de données CMS utilisés par la présente Recommandation et par d'autres spécifications de télébiométrie. Ces dernières peuvent faire référence à la présente Annexe dans le cadre de leur utilisation de CMS.

L'objectif ici n'est pas de spécifier une implémentation qui ne serait pas compatible avec les RFC de l'IETF, mais seulement d'examiner les aspects de la syntaxe CMS qui sont pertinents pour les spécifications de télébiométrie. L'Appendice I définit un module ASN.1 informel qui rend compte de l'utilisation de la syntaxe CMS en télébiométrie.

CMS définit différents types de contenu à utiliser à diverses fins. Les spécifications de télébiométrie utilisent le type de contenu **signedData** pour assurer l'authentification et l'intégrité, et le type de contenu **envelopedData** pour assurer le chiffrement et donc la confidentialité; elles utilisent également le type de contenu **ct-authEnvelopedData**. Le type de contenu **signedData** est employé lorsqu'une signature numérique est nécessaire, tandis que le type de contenu **envelopedData** est utilisé lorsque la confidentialité est requise. Lorsqu'on y a recours, une instance du type de contenu **envelopedData** est encapsulée dans une instance du type de contenu **signedData**. Le type de contenu **ct-authEnvelopedData** est utilisé lorsqu'une tâche est constituée de plusieurs messages.

Les aspects des types de contenu susmentionnés ne sont pas tous utilisés par les spécifications de télébiométrie. C'est pourquoi la présente Annexe définit un profil pour l'utilisation de la syntaxe CMS en télébiométrie. Pour faciliter la consultation, les aspects pertinents de CMS sont repris ci-après.

Une instance d'un type de contenu défini par les spécifications télébiométriques est encapsulée dans une instance du type de contenu **envelopedData**, si la confidentialité doit être respectée; sinon, elle est encapsulée dans une instance du type de contenu **signedData**. Il est également possible d'intégrer une telle instance dans une instance du type de contenu **ct-authEnvelopedData**.

D'après le document [IETF RFC 6268], un type de contenu est défini au moyen de la classe d'objets informationnels suivante:

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

La classe d'objets informationnels **CONTENT-TYPE** est équivalente à la classe d'objets informationnels ASN.1 prédéfinie **TYPE-IDENTIFIER**. Un objet informationnel **CONTENT-TYPE** est utilisé pour relier le type de contenu identifié par un identificateur d'objet à la syntaxe abstraite du contenu.

Le type de données **ContentInfo** décrit ci-dessous définit la syntaxe générale d'un type de contenu:

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

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

Les types de contenu pris en charge sont: **signedData**, **envelopedData**, **ct-authEnvelopedData** et l'ensemble des types de contenu définis par une spécification télébiométrique particulière.

La syntaxe CMS exige que pour tout type de données, il soit précisé la version de CMS utilisée. Les versions sont définies comme suit:

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

Le document [IETF RFC 6268] définit le type de données paramétré suivant, qui est utilisé dans l'ensemble des spécifications:

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

B.2 Utilisation du type de contenu signedData

Le type de contenu suivant est décrit au § 5 du document [IETF RFC 5652]. Moyennant un léger changement de notation pour l'adapter au domaine de la télébiométrie, ce type de contenu est défini comme suit:

```
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 – Le document [IETF RFC 6268] utilise une version légèrement modifiée du type de données **AlgorithmIdentifier**. Cela étant, le présent profil utilise le type de données **AlgorithmIdentifier** tel qu'il est défini dans la Recommandation [UIT-T X.509].

Le composant **version** doit prendre la valeur **v3** conformément au § 5.1 du document [IETF RFC 5652].

Le composant **digestAlgorithms** consiste en un seul élément qui précise l'algorithme de hachage utilisé parmi l'ensemble des algorithmes de hachage possibles, tel que défini par la spécification télébiométrique applicable.

La définition suivante permet d'inclure n'importe quel algorithme de hachage.

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

NOTE 2 – Le présent profil n'impose pas l'utilisation d'un algorithme de hachage particulier. Dans les spécifications de référence ou les accords d'implémentation, les points peuvent être remplacés par un ensemble d'algorithmes de hachage à prendre en charge dans un environnement particulier.

NOTE 3 – CMS permet d'utiliser plusieurs signatures numériques et donc plusieurs algorithmes de hachage. Le recours à de multiples signatures numériques n'est pas pertinent pour les spécifications télébiométriques.

Le composant **encapContentInfo** contient une instance du type de données suivant:

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

```
    (CONTAINING CONTENT-TYPE.&Type({envelopedData, ...}
     {@eContentType})) OPTIONAL }
```

Ce type de données est constitué des composants suivants:

a) Le composant **eContentType** contient l'identificateur d'objets qui identifie le type de contenu encapsulé. Si un chiffrement doit être appliqué, ce composant contient l'identité du type de contenu **envelopedData**. Dans le cas contraire, il contient l'un des types de contenu définis par la spécification télébiométrique pertinente;

b) Le composant **econtent** contient le contenu encapsulé proprement dit, intégré dans une chaîne d'octets. Il doit toujours être présent.

NOTE 4 – Ce composant est défini comme étant optionnel. Cela étant, tous les types de contenu pertinents ont un contenu défini.

Le composant **certificates** contient les certificats de clé publique qui sont suffisants pour établir un seul chemin de certification, tel que spécifié par le type de données **PkiPath** défini dans la Recommandation [UIT-T X.509].

Le composant **crls** ne s'applique pas aux spécifications télébiométriques; il est donc absent.

Le composant **signerInfos** contient une instance du type de données **SignerInfos** :

```
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 ::= {...}
```

Le présent profil ne prenant en charge qu'un seul signataire, une instance du type de données **SignerInfos** contient un élément et un seul. Le type de données **SignerInfo** contient les composants suivants:

a) Le composant **version** prend la valeur **v1** conformément au document [IETF RFC 5652].

b) Le composant **sid** identifie le certificat de clé publique d'entité finale du signataire et contient une instance du type de données **SignerIdentifier**. Ce type de données peut prendre deux formes possibles:

– la forme **issuerAndSerialNumber** identifie le certificat de clé publique d'entité finale en indiquant le nom distinctif de la CA émettrice ainsi que le numéro de série de ce dernier. Cette forme doit toujours être choisie;

– la forme **subjectKeyIdentifier** ne doit jamais être retenue.

c) Le composant **digestAlgorithm** contient la même valeur que celle utilisée dans le composant **digestAlgorithms** du type de données **SignerInfo**.

d) Le composant **signedAttrs** contient une liste d'attributs signés. Le document [IETF RFC 5652] impose qu'y figurent au minimum des instances des types d'attribut **contentType** et **messageDigest**. Le présent profil n'exige pas d'inclure d'autres attributs dans cette liste, mais des spécifications faisant référence pourront éventuellement la compléter.

e) Le composant **signatureAlgorithm** contient l'algorithme de signature utilisé pour créer la signature numérique figurant dans le composant **signature**;

NOTE 5 – Le présent profil n'impose pas l'utilisation d'un algorithme de signature particulier. Dans les spécifications de référence ou les accords d'implémentation, les points peuvent être remplacés par un ensemble d'algorithmes de signature à prendre en charge pour une spécification télébiométrique particulière.

f) Les composants **signature** et **unsignedAttrs** tels que spécifiés dans le document [IETF RFC 5652].

B.3 Utilisation du type de contenu envelopedData

B.3.1 Généralités

Le type de contenu **envelopedData** permet d'effectuer le chiffrement des données. Pour cela, il est nécessaire de créer des clés symétriques partagées. Le document [IETF RFC 5652] fournit plusieurs techniques pour créer des clés symétriques. Le présent profil exige l'utilisation de la technique d'agrément de clés appelée "méthode d'agrément de clés Diffie-Hellman (DH)". Cette méthode est décrite dans le document [IETF RFC 2631] dans le cas de la technique à courbe elliptique. Le document [IETF RFC 5753] fournit les spécifications à utiliser pour les techniques à courbe elliptique.

La méthode DH produit un secret partagé qui peut être utilisé comme données de calcul de clé pour générer des clés symétriques partagées. Le présent profil prend en charge deux modes de fonctionnement DH: le mode éphémère-statique et le mode statique-statique.

Dans le mode éphémère-statique, le destinataire doit disposer d'un certificat de clé publique avec une clé publique DH certifiée par la CA émettrice. Ce certificat de clé publique doit être mis à disposition de l'émetteur. Ce dernier crée, pour chaque message qu'il envoie, une nouvelle paire de clés DH. Ainsi, le secret partagé est différent pour chaque message envoyé.

Dans le mode statique-statique, chaque entité communicante doit disposer d'un certificat de clé publique DH certifié. Pour éviter que le secret partagé ne soit identique à chaque message, l'émetteur doit fournir des données de calcul de clé d'utilisateur aléatoires, et donc différentes, pour chaque message.

Le présent profil exige que le mode éphémère-statique soit pris en charge.

Pour ces deux méthodes, chaque entité d'une communication doit connaître le certificat de clé publique d'entité finale DH de son interlocuteur, puisque les communications sont bidirectionnelles.

Le type de contenu suivant est décrit au § 6 du document [IETF RFC 5652] et actualisé dans le document [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 }
```

Le composant **EnvelopedData** est constitué des éléments suivants:

a) Le composant **version** doit, en vertu du document [IETF RFC 5652], prendre la valeur **v2** si le composant **unprotectedAttrs** est présent. Sinon, il doit être égal à **v0**.

b) Le composant **originatorInfo** ne doit pas être présent.

c) Le composant **recipientInfos** contient une instance du type de données **accessService**, tel que défini au § B.3.2.

d) Le composant **encryptedContentInfo** contient une instance du type de données **EncryptedContentInfo**, tel que défini au § B.3.4.

e) Le composant **unprotectedAttrs** est requis s'il est attendu que le prochain élément à émettre dans la direction en question est une instance du type de contenu **ct-authEnvelopedData**. Dans le cas contraire, il peut être omis.

B.3.2 Informations concernant le destinataire

Le type de données **RecipientInfos** peut accepter plusieurs instances, mais aux fins du présent profil, il sera limité à une seule. Une instance du type de données **RecipientInfo** spécifie différentes possibilités d'établir un secret partagé entre deux interlocuteurs.

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

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

Le présent profil n'utilise que deux des formes possibles de **RecipientInfo** définies dans [IETF RFC 5652]:

a) La forme **kari** est le seul choix possible pour le type de contenu **envelopedData**. Le type de données **KeyAgreeRecipientInfo** fournit les informations nécessaires à la mise en place d'un secret partagé tel que défini au § B.3.3.

b) La forme **kekri** est le seul choix possible pour le type de contenu **ct-authEnvelopedData**. Le type de données **KEKRecipientInfo** fournit les informations nécessaires à la mise en place d'un secret partagé tel que défini au § B.4.

B.3.3 Agrément de clé

```
KeyAgreeRecipientInfo ::= SEQUENCE {
  version             six   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
```

Le type de données **KeyAgreeRecipientInfo** contient les composants suivants:

a) Le composant **version** doit prendre la valeur **v3** conformément au document [IETF RFC 5652].

b) Le composant **originator** contient une instance du type de données **OriginatorIdentifierOrKey** avec les formes possibles suivantes:

- La forme **issuerAndSerialNumber** est retenue si la méthode statique-statique est utilisée. Elle contient alors une instance du type de données **IssuerAndSerialNumber**. Ce type de données identifie le certificat de clé publique DH de l'émetteur:
  - i) Le composant **issuer** contient le nom distinctif de la CA chargée de délivrer le certificat de clé publique et est égal au composant **issuer** du certificat en question;
  - ii) Le composant **serialNumber** est égal au composant **serialNumber** de ce certificat de clé publique;
- La forme **subjectKeyIdentifier** ne doit jamais être retenue.
- La forme **originatorKey** est retenue si la méthode éphémère-statique DH est utilisée. Elle contient alors une instance du type de données **OriginatorPublicKey** avec les formes possibles suivantes:
  - i) Le composant **algorithm** contient une référence à l'algorithme de clé publique DH utilisé.

NOTE 1 – Le présent profil n'impose pas l'utilisation d'un algorithme de clé publique DH particulier. Dans les spécifications de référence ou les accords d'implémentation, les points peuvent être remplacés par un ensemble d'algorithmes de clé publique DH à prendre en charge dans un environnement particulier.

ii) Le composant **publicKey** contient la clé publique DH générée par l'émetteur. L'émetteur génère une nouvelle paire de clés DH à chaque utilisation de ce type de contenu.

Dès lors qu'il détient la clé privée locale et la clé publique du destinataire, l'émetteur peut générer le secret partagé. Le destinataire génère un secret partagé identique à partir de sa clé privée et de la clé publique de l'émetteur fourni dans le type de données **OriginatorPublicKey** ou **IssuerAndSerialNumber**.

c) Le composant **ukm** est présent lorsque la méthode statique-statique est utilisée. Il contient alors une instance du type de données **UserKeyingMaterial**.

A partir du secret partagé, de la valeur du composant **ukm** (si applicable) et de quelques autres données telles que spécifiées dans [IETF RFC 2631], les deux parties génèrent ce que l'on appelle la clé de chiffrement de clé (KEK). Cette clé est ensuite utilisée pour chiffrer la clé de chiffrement de contenu (CEK) générée par l'émetteur. Cette technique est appelée "enveloppement de clé" (*key wrapping*).

d) Le composant **keyEncryptionAlgorithm** précise l'algorithme d'enveloppement de clé et contient une instance du type de données **KeyEncryptionAlgorithmIdentifier**;

NOTE 2 – Le présent profil n'impose pas d'ensemble particulier d'algorithmes d'enveloppement de clé. De nouveaux algorithmes pourront être définis à l'avenir. Les algorithmes d'enveloppement de clé pour la norme de cryptage évoluée (AES) sont définis dans le document [IETF RFC 3394]. Dans les spécifications de référence ou les accords d'implémentation, les points peuvent être remplacés par un ensemble d'algorithmes de d'enveloppement à prendre en charge dans un environnement particulier.

e) Le composant **recipientEncryptedKeys** contient une instance du type de données **RecipientEncryptedKeys**. Cette instance est composée d'un seul élément, à savoir une instance unique du type de données **RecipientEncryptedKey**. Ce type de données est constitué des deux composants suivants:

– Le composant **rid** contient l'identification du destinataire par son certificat de clé publique d'entité finale.

– Le composant **encryptedKey** contient la clé CEK chiffrée utilisée pour chiffrer le contenu, comme indiqué au point c).

B.3.4 Réutilisation de clés de chiffrement de contenu CMS

S'il est envisagé de réutiliser la clé CEK pour une instance ultérieure du type de contenu **ct-authEnvelopedData** tel que spécifié dans [IETF RFC 3185], les informations de référence appropriées doivent être stockées dans les attributs non protégés examinés au § B.3.1. Ces informations sont conservées par les deux parties. S'il faut assurer un haut niveau de sécurité, l'attribut de type **aa-CEKMaxDecrypts** doit prendre la valeur '1' ou être omis.

B.3.5 Informations de contenu chiffrées

Une instance du type de données **EncryptedContentInfo** contient le contenu encapsulé chiffré.

```
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
```

Le composant **encryptedContentInfo** du type de données **EnvelopedData** contient une instance du type de données **EncryptedContentInfo**:

a) Le composant **contentType** contient le type de contenu pour le contenu chiffré. Les types de contenu possibles sont ceux pour lesquels le chiffrage est optionnel.

b) Les composants **contentEncryptionAlgorithm** et **encryptedContent** tels qu'exigés par [IETF RFC 5652].

B.4 Utilisation du type de contenu Authenticated-Enveloped-Data

B.4.1 Généralités

Comme cela est spécifié dans la Recommandation [UIT-T X.1080.1], les protocoles applicables à la télébiométrie consistent en règle générale en un échange préliminaire destiné à établir la session, suivi de multiples échanges d'informations, et enfin de la terminaison de la session. Dans ce type d'environnement, il n'est pas forcément nécessaire de générer une nouvelle clé de chiffrement de clé à chaque message.

Le document [IETF RFC 5083] spécifie un type de contenu dénommé **ct-authEnvelopedData** qui ne figure pas dans [IETF RFC 5652]. Ce type de contenu permet d'utiliser des techniques de chiffrement authentifié efficaces. Le présent profil utilise les algorithmes AES-GCM définis dans [IETF RFC 5084] et réutilise la clé CEK définie dans [IETF RFC 3185]. Pour de plus amples informations, on se reportera à ces spécifications.

Le type de contenu **ct-authEnvelopedData** est défini comme suit:

```
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 }
```

Le type de données **AuthEnvelopedData** contient les composants suivants:

a) Le composant **version** doit prendre la valeur **v0** conformément au document [IETF RFC 5083].

b) Le composant **originatorInfo** ne doit pas être présent.

c) Le composant **recipientInfos** contient une instance du type de données **RecipientInfos**. Ce type de données est décrit au § B.3.2. Outre la forme **kari**, la forme **kekri** est pertinente pour ce type de contenu. Lorsque la forme **kekri** est retenue, elle contient une instance du type de données **KEKRecipientInfo**, tel que défini au § B.4.2.

d) Le composant **authEncryptedContentInfo** contient une instance du type de données **EncryptedContentInfo**, tel que défini au § B.3.5.

e) Le composant **authAttrs**, lorsqu'il est présent, contient un ensemble d'attributs qui doivent être protégés par authentification.

f) Le composant **mac** contient le code d'authentification de message généré (MAC).

g) Le composant **unauthAttrs** contient des attributs du même type que celui indiqué au point e) du § B.3.1. S'il est connu que l'instance du type de contenu est la dernière de la session pour le sens concerné, ce composant peut être omis.

B.4.2 Informations concernant le destinataire de la clé KEK

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

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

Le type de données **KEKRecipientInfo** contient les composants suivants:

a) Le composant **version** doit prendre la valeur **v4** conformément au document [IETF RFC 5652].

b) Le composant **kekid** contient une instance du type de données **KEKIdentifier**, lequel se compose des éléments suivants:

- le composant **keyIdentifier** contient l'identificateur de la clé CEK récupéré lors d'un échange précédent, comme spécifié au § B.3.4;
- les autres composants ne sont pas nécessaires;

c) Le composant **keyEncryptionAlgorithm** contient l'algorithme d'enveloppement indiqué au point d) du § B.3.3. Pour une session télébiométrique donnée, il est recommandé d'utiliser le même algorithme d'enveloppement pour toutes les instances de contenu.

B.5 Attributs

Les types d'attribut suivants sont définis dans [IETF RFC 5652]. Les instances de ces types d'attribut sont destinées à être utilisées comme des attributs signés.

```
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 }
```

Les types d'attribut suivants sont définis dans [IETF RFC 3185]. Les instances de ces types d'attribut peuvent être utilisées comme des attributs signés.

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

```
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 Codes d'erreur de la syntaxe de message cryptographique

Le document [b-IETF RFC 7191] fournit une liste des codes d'erreur couvrant tous les usages possibles de la syntaxe CMS. Un sous-ensemble de ces codes applicable à la télébiométrie est donné ci-dessous. On trouvera une description des codes dans le document [b-IETF RFC 7191].

Lorsque le document [b-IETF RFC 7191] fait référence à un certificat, il s'agit d'une référence au certificat de clé publique utilisé pour les types de contenu CMS définis.

```
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) }
```

Annexe C

Spécification formelle des protocoles de déclaration et d'attribution de privilèges

(Cette annexe fait partie intégrante de cette Recommandation.)

```
Pbact-access { joint-iso-itu-t(2) telebiometrics(42) e-health-protocol(3)
   modules(0) pbact-access(6) 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) th(3) 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-pbact              OBJECT IDENTIFIER ::=
  { joint-iso-itu-t(2) telebiometrics(42) e-health-protocol(3) pbact(20) }
id-pbactmodule        OBJECT IDENTIFIER ::= { id-pbact module(0) }
id-pbactCont          OBJECT IDENTIFIER ::= { id-pbact cmsCont(1) }
id-pbactPrivAttr      OBJECT IDENTIFIER ::= { id-pbact prAttr(2) }

-- Content types

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


-- Artribute types for carring privilege definitions

id-at-accessService OBJECT IDENTIFIER ::= { id-pbactPrivAttr 1 }

END
```

Appendice I

Spécification informelle pour le profil de syntaxe de message cryptographique

(Cet Appendice ne fait pas partie intégrante de la présente Recommandation.)

Une implémentation qui ne prendrait en charge que le module **CmsTelebiometric** ne serait pas conforme à la spécification CMS de l'IETF et ne doit donc figurer dans aucune spécification d'implémentation. La description de ce module n'est donnée qu'à titre informatif et aux fins du contrôle de cohérence.

```
CmsTelebiometric { joint-iso-itu-t(2) telebiometrics(42) th(3) 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 ::= INTEGERa

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
```

Bibliographie

[b-UIT-T X.841] Recommandation UIT-T X.841 (2000) | ISO/CEI 15816:2002, *Technologies de l'information – Techniques de sécurité – Objets informationnels de sécurité pour le contrôle d'accès*.

[b-CEI 62351-8] CEI/TS 62351-8:2011, *Gestion des systèmes de puissance et échanges d'informations associés – Sécurité des communications et des données – Partie 8: Gestion de clé de cybersécurité des équipements de système de puissance*.

[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.*

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série D	Principes de tarification et de comptabilité et questions de politique générale et d'économie relatives aux télécommunications internationales/TIC
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Environnement et TIC, changement climatique, déchets d'équipements électriques et électroniques, efficacité énergétique; construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	Gestion des télécommunications y compris le RGT et maintenance des réseaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation et mesures et tests associés
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	**Réseaux de données, communication entre systèmes ouverts et sécurité**
Série Y	Infrastructure mondiale de l'information, protocole Internet, réseaux de prochaine génération, Internet des objets et villes intelligentes
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Imprimé en Suisse
Genève, 2017