Groupe de travail Réseau
Request for Comments : 5652
STD70
RFC rendue obsolète : 3852
Catégorie : Norme

R. Housley, Vigil Security

septembre 2009

Traduction Claude Brière de L'Isle

Syntaxe de message cryptographique (CMS)

Résumé
Le présent document décrit la syntaxe de message cryptographique (CMS). Cette syntaxe est utilisée pour la signature numérique, le résumé, l'authentification, ou le chiffrement de contenus arbitraires de message.

Statut de ce mémoire
Le présent document spécifie un protocole Internet en cours de normalisation pour la communauté de l'Internet, et appelle à des discussions et des suggestions pour son amélioration. Prière de se reporter à l'édition actuelle du STD 1 "Normes des protocoles officiels de l'Internet" pour connaître l'état de normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.
(La présente traduction incorpore l'errata 2026)

Table des matières

1. Introduction

Le présent document décrit la syntaxe de message cryptographique (CMS, *Cryptographic Message Syntax*). Cette syntaxe est utilisée pour la signature numérique, le résumé, l'authentification ou le chiffrement de contenus arbitraires de message.

La CMS décrit une syntaxe d'encapsulation pour la protection des données. Elle prend en charge les signatures numériques et le chiffrement. La syntaxe permet plusieurs encapsulations ; une enveloppe d'encapsulation peut être incorporée dans une autre. De même, une partie peut signer numériquement certaines données précédemment encapsulées. Elle permet aussi que des attributs arbitraires, tels que l'heure de signature, soient signés en même temps que le contenu du message, et elle permet que d'autres attributs comme des contreseings soient associés à une signature.

La CMS peut accepter diverses architectures de gestion de clé fondée sur le certificat, comme celle définie par le groupe de travail PKIX (Infrastructure de clé publique utilisant X.509) [RFC5280].

Les valeurs de la CMS sont générées à l'aide de l'ASN.1 [X.208-88], en utilisant les codages BER (Règles de codage de base) [X.209-88]. Les valeurs sont normalement représentées comme des chaînes d'octets. Alors que de nombreux systèmes sont capables de transmettre fiablement des chaînes d'octets arbitraires, il est bien connu que de nombreux systèmes de messagerie électronique ne le sont pas. Le présent document ne vise pas les mécanismes de codage des chaînes d'octets pour une transmission fiable dans de tels environnements.

1.1 Évolution de la CMS

La CMS est dérivée de PKCS n° 7 version 1.5, qui est exposée dans la [RFC2315]. PKCS n° 7 version 1.5 a été développée en dehors de l'IETF ; elle a été à l'origine publiée comme note technique des Laboratoires RSA en novembre 1993. L'IETF a depuis pris la responsabilité du développement et de la maintenance de la CMS. Aujourd'hui, plusieurs importants protocoles en cours de normalisation de l'IETF font usage de la CMS.

La présente section décrit les changements que l'IETF a apporté à la CMS dans chacune des versions publiées.

1.1.1 Changements par rapport à PKCS n° 7 version 1.5

La [RFC2630] était la première version de la CMS sur la voie de la normalisation par l'IETF. Chaque fois que possible, la rétro compatibilité avec PKCS n° 7 version 1.5 a été préservée ; cependant, des changements ont été apportés pour s'accommoder du transfert de certificat d'attribut version 1 et pour prendre en charge la gestion de clé indépendante de l'algorithme. PKCS n° 7 version 1.5 n'incluait la prise en charge que du transport de clé. La RFC 2630 ajoute la prise en charge des techniques d'accord de clé et de la clé de chiffrement de clé symétrique distribuée à l'avance.

1.1.2 Changements depuis la RFC 2630

La [RFC3369] rend obsolètes les [RFC2630] et [RFC3211]. La gestion de clé fondée sur le mot de passe est incluse dans la spécification de la CMS, et un mécanisme d'extension pour la prise en charge de nouveaux schémas de gestion de clé est spécifié sans autre changement à la CMS. La rétro compatibilité avec les RFC2630 et RFC3211 est préservée ; cependant, le transfert de certificat d'attribut version 2 est ajouté, et l'utilisation du certificat d'attribut version 1 est déconseillée.

Les signatures des extensions de messagerie Internet sécurisée/multi-objets (S/MIME) v2 [RFC2311], qui se fondent sur PKCS n° 7 version 1.5, sont compatibles avec les signatures S/MIME v3 [RFC2633] et les signatures S/MIME v3.1 [RFC3851]. Cependant, il y a quelques problèmes subtils de compatibilité avec les signatures fondées sur PKCS n° 7 v.1.5. Ces questions sont exposées au paragraphe 5.2.1. Ces problèmes demeurent avec la version actuelle de la CMS.

Les algorithmes cryptographiques spécifiques ne sont pas discutés dans le présent document, mais ils l'étaient dans la RFC2630. La discussion des algorithmes cryptographiques spécifiques a été déplacée dans un document distinct, la [RFC3370]. La séparation de la spécification du protocole et de l'algorithme permet à l'IETF de mettre à jour chaque document de façon indépendante. La présente spécification n'exige la mise en œuvre d'aucun algorithme particulier. Les protocoles qui s'appuient sur la CMS sont plutôt supposés choisir les algorithmes appropriés à leur environnement. Les algorithmes peuvent être choisis à partir de la [RFC3370] ou ailleurs.

1.1.3 Changements par rapport à la RFC3369

La [RFC3852] rend obsolète la [RFC3369]. Comme exposé au paragraphe précédent, la RFC 3369 introduisait un mécanisme d'extension pour la prise en charge des schémas de gestion de clé sans autre changement à la CMS. La RFC3852 introduit un mécanisme d'extension similaire pour prendre en charge des formats de certificat supplémentaires et des formats d'information d'état de révocation sans autre changement à la CMS. Ces extensions sont principalement documentées aux paragraphes 10.2.1 et 10.2.2. La rétro compatibilité avec les versions antérieures de la CMS est préservée.

L'utilisation des numéros de version est décrite au paragraphe 1.3.

Depuis la publication de la RFC3369, quelques erreurs ont été notées. Les errata sont accessibles sur le site de l'éditeur des RFC. Ces erreurs ont été corrigées dans le présent document.

Le texte du paragraphe 11.4 qui décrit l'attribut non signé de contreseing a été précisé. Heureusement, le texte révisé est plus clair sur la portion de la signature SignerInfo que celui qui traite du contreseing.

1.1.4 Changements par rapport à la RFC 3852

Le présent document rend obsolète la [RFC3852]. La principale raison de la publication de ce document est de faire avancer la CMS le long de l'échelle de maturité des normes.

Le présent document comporte les éclaircissements qui ont été à l'origine publiés dans la [RFC4853] concernant le traitement approprié du type de contenu protégé SignedData lorsque plus d'une signature numérique est présente.

Depuis la publication de la RFC3852, quelques erreurs ont été notées. Des errata figurent sur le site de l'éditeur des RFC. Ces erreurs ont été corrigées dans le présent document.

1.2 Terminologie

Dans ce document, les mots clés DOIT, NE DOIT PAS, EXIGE, DEVRAIT, NE DEVRAIT PAS, RECOMMANDE, PEUT et FACULTATIF sont à interpréter comme décrit dans la [RFC2119].

1.3 Numéros de version

Chacune des structures de données majeures comporte un numéro de version comme premier élément de la structure des données. Les numéros de version sont destinés à éviter les erreurs de décodage ASN.1. Certaines mises en œuvre ne vérifient pas le numéro de version avant de tenter le décodage, et si une erreur de décodage survient, le numéro de version est alors vérifié au titre de la routine de traitement d'erreur. C'est une approche raisonnable ; elle place le traitement des erreurs en dehors du chemin rapide normal. Cette approche pardonne aussi l'utilisation de numéros de version incorrects par l'envoyeur.

La plupart des numéros de version initiaux ont été alloués dans PKCS n° 7 v.1.5. D'autres ont été alloués lorsque la structure a été initialement créée. Chaque fois qu'une structure est mise à jour, un numéro de version plus élevé est alloué. Cependant, pour assurer une interopérabilité maximale, le numéro de version plus élevé n'est utilisé que lorsque la nouvelle caractéristique syntaxique est employée. C'est-à-dire qu'on utilise le plus faible numéro de version qui prend en charge la syntaxe générée.

2. Présentation générale

La CMS est assez générale pour prendre en charge de nombreux types de contenu différents. Le présent document définit un contenu de protection, ContentInfo. ContentInfo encapsule un seul type de contenu identifié, et le type identifié peut fournir d'autres encapsulations. Le présent document définit six types de contenu : data, signed-data, enveloped-data, digested-data, encrypted-data, et authenticated-data. Des types de contenu supplémentaires peuvent être définis en dehors du présent document.

Une mise en œuvre qui se conforme à la présente spécification DOIT mettre en œuvre le contenu de protection, ContentInfo, et DOIT mettre en œuvre les types de contenu data, signed-data, et enveloped-data. Les autres types de contenu PEUVENT être mis en œuvre.

Selon la philosophie générale de la conception, chaque type de contenu permet un traitement en un seul passage en utilisant un codage de longueur indéfinie des règles de codage de base (BER, *Basic Encoding Rules*). Le fonctionnement en un seul passage est particulièrement utile si le type de contenu est grand, mémorisé sur des bandes magnétiques, ou est "pompé" d'un autre processus. Le fonctionnement en un seul passage a un inconvénient significatif : il est difficile d'effectuer des opérations de codage en utilisant le codage des règles de codage distinctives (DER, *Distinguished Encoding Rules*) de [X.509-88] en un seul passage car les longueurs des divers composants peuvent n'être pas connues à l'avance. Cependant, des attributs signés au sein du type de contenu signed-data et des attributs authentifiés au sein du type de contenu authenticated-data ont besoin d'être transmis en forme DER pour s'assurer que le receveur peut vérifier un contenu qui comporte un ou plusieurs attributs non reconnus. Les attributs signés et les attributs authentifiés sont les seuls types de données utilisés dans la CMS qui exigent le codage en DER.

3. Syntaxe générale

L'identifiant d'objet suivant identifie le type d'information de contenu :

IDENTIFIANT D'OBJET id-ct-contentInfo ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
smime(16) ct(1) 6 }

La CMS associe un identifiant de type de contenu à un contenu. La syntaxe DOIT avoir le type ASN.1 ContentInfo :

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content [0] EXPLICITE TOUT DEFINI PAR contentType }

ContentType ::= IDENTIFIANT D'OBJET
```

Les champs de ContentInfo ont la signification suivante :

contentType indique le type du contenu associé. C'est un identifiant d'objet ; c'est une chaîne unique d'entiers allouée par une autorité qui définit le type de contenu.

content est le contenu associé. Le type de contenu peut être déterminé de façon univoque par le contentType. Les types de contenu pour data, signed-data, enveloped-data, digested-data, encrypted-data, et authenticated-data sont définis dans le présent document. Si des types de contenu supplémentaires sont définis dans d'autres documents, le type ASN.1 défini NE DEVRAIT PAS être un type CHOIX.

4. Type de contenu Data

L'identifiant d'objet suivant identifie le type de contenu data :

IDENTIFIANT D'OBJET id-data ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 1 }

Le type de contenu data est destiné à se référer à des chaînes d'octets arbitraires, telles que des fichiers de texte en ASCII ; l'interprétation en est laissée à l'application. Il n'est pas nécessaire que de telles chaînes aient une structure interne (bien qu'elles puissent avoir leur propre définition ASN.1 ou une autre structure).

S/MIME utilise id-data pour identifier le contenu codé en MIME. L'utilisation de cet identifiant de contenu est spécifiée dans la [RFC2311] pour S/MIME v2, dans la [RFC2633] pour S/MIME v3, et dans la [RFC3851] pour S/MIME v3.1.

Le type de contenu data est généralement encapsulé dans les types de contenu signed-data, enveloped-data, digested-data, encrypted-data, ou authenticated-data.

5. Type de contenu Signed-data

Le type de contenu signed-data *(données signées)* consiste en un contenu de type quelconque et de zéro, une ou plusieurs valeurs de signature. Un nombre quelconque de signataires peuvent signer en parallèle tout type de contenu.

L'application normale du type de contenu signed-data représente une signature numérique du signataire sur le contenu du type de contenu data. Une autre application typique dissémine les certificats et les listes de révocation de certificat (CRL, *Certificate Revocation List*).

Le processus de construction des données signées comporte les étapes suivantes :

1. Pour chaque signataire, un résumé de message, ou une valeur de hachage, est calculé sur le contenu avec un algorithme de résumé de message spécifique du signataire. Si le signataire signe des informations autres que le contenu, le résumé de message du contenu et les autres informations sont résumés avec l'algorithme de résumé de message du signataire (voir au paragraphe 5.4) et le résultat devient le "résumé de message".

2. Pour chaque signataire, le résumé de message est signé numériquement en utilisant la clé privée du signataire.

3. Pour chaque signataire, la valeur de la signature et les autres informations spécifiques du signataire sont collectées dans une valeur SignerInfo, définie au paragraphe 5.3. Les certificats et CRL pour chaque signataire, et ceux qui ne correspondent à aucun signataire, sont collectés à cette étape.

4. Les algorithmes de résumé de message pour tous les signataires et les valeurs de SignerInfo pour tous les signataires sont collectés avec le contenu en une valeur SignedData, définie au paragraphe 5.1.

Un receveur calcule de façon indépendante le résumé de message. Ce résumé de message et la clé publique du signataire sont utilisés pour vérifier la valeur de la signature. La clé publique du signataire est référencée d'une des deux façons suivantes. Elle peut être référencée par le nom distinctif de l'émetteur avec un numéro de série spécifique de l'émetteur pour identifier de façon univoque le certificat qui contient la clé publique. Autrement, elle peut être référencée par un identifiant de clé de sujet, qui s'accommode aussi bien de clés publiques certifiées que non certifiées. Bien qu'il ne soit pas exigé, le certificat du signataire peut être inclus dans le champ des certificats SignedData.

Lorsque il y a plus d'une signature, la réussite de la validation d'une signature associée à un certain signataire est normalement traitée comme une signature réussie par ce signataire. Cependant, il existe certains environnements d'application où d'autres règles sont nécessaires. Une application qui utilise une règle autre que celle d'une signature valide pour chaque signataire doit spécifier ces règles. De même, lorsque une simple correspondance de l'identifiant du signataire n'est pas suffisante pour déterminer si les signatures ont été générées par le même signataire, la spécification de l'application doit décrire comment déterminer quelles signatures ont été générées par un certain signataire. La prise en charge de différentes communautés de receveurs est la principale raison pour laquelle les signataires choisissent d'inclure plus d'une signature. Par exemple, le type de contenu signed-data peut comporter des signatures générées avec l'algorithme de signature RSA et avec l'algorithme de signature numérique à courbe elliptique (ECDSA, *Elliptic Curve Digital Signature Algorithm*). Cela permet aux receveurs de vérifier les signatures associées à l'un ou l'autre algorithme.

La présente section est divisée en six parties. La première décrit le type de niveau supérieur SignedData, la seconde décrit EncapsulatedContentInfo, la troisième décrit le type d'information par signataire SignerInfo, et les quatrième, cinquième et sixième décrivent les processus, respectivement, de calcul de résumé de message, de génération des signatures, et de vérification des signatures.

5.1 Type SignedData

L'identifiant d'objet suivant identifie le type de contenu signed-data :

IDENTIFIANT D'OBJET id-signedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }

Le type de contenu signed-data devra avoir le type ASN.1 SignedData :

```
SignedData ::= SEQUENCE {
    version                 CMSVersion,
    digestAlgorithms        DigestAlgorithmIdentifiers,
    encapContentInfo        EncapsulatedContentInfo,
    certificates [0]        IMPLICITE CertificateSet OPTIONNEL,
    crls [1]                IMPLICITE RevocationInfoChoices OPTIONNEL,
    signerInfos             SignerInfos }

DigestAlgorithmIdentifiers ::= ENSEMBLE DE DigestAlgorithmIdentifier

SignerInfos ::= ENSEMBLE DE SignerInfo
```

Les champs du type SignedData ont la signification suivante :

version est le numéro de version de la syntaxe. La valeur appropriée dépend des certificats, du eContentType, et des SignerInfo. La version DOIT être allouée comme suit :

```
SI ((un certificat est présent) ET
    (un certificat avec un type autre est présent)) OU
    ((crls est présent) ET
    (un crls avec un type autre est présent))
ALORS version DOIT être 5
AUTREMENT
    SI (un certificat est présent) ET
        (un certificat d'attribut de version 2 est présent)
    ALORS version DOIT être 4
    AUTREMENT
        SI ((un certificat est présent) ET
            (un certificat d'attribut de version 1 est présent)) OU
            (une structure SignerInfo est de version 3) OU(encapContentInfo eContentType est autre que id-data)
        ALORS version DOIT être 3
        AUTREMENT version DOIT être 1
```

digestAlgorithms est une collection d'identifiants d'algorithme de résumé de message. Il PEUT y avoir n'importe quel nombre d'éléments dans la collection, y compris zéro. Chaque élément identifie l'algorithme de résumé de message, ainsi que tous paramètres associés, utilisé par un ou plusieurs signataires. La collection est destinée à faire la liste des algorithmes de résumé de message employés par tous les signataires, dans un ordre quelconque, pour faciliter la vérification de signature en un seul passage. Les mises en œuvre PEUVENT échouer à valider les signatures qui utilisent un algorithme de résumé qui n'est pas inclus dans cet ensemble. Le processus de résumé de message est décrit au paragraphe 5.4.

encapContentInfo est le contenu signé, consistant en un identifiant de type de contenu et en ce contenu lui-même. Les détails du type EncapsulatedContentInfo sont exposés au paragraphe 5.2.

certificates est une collection de certificats. Il est destiné à ce que l'ensemble des certificats soit suffisant pour contenir les chemins de certification à partir d'une "racine" reconnue ou d'une "autorité de certification de niveau supérieur" pour tous les signataires dans le champ signerInfos. Il peut y avoir plus de certificats que nécessaire, et il peut y avoir des certificats suffisants pour contenir des chemins de certification provenant de deux ou plus autorités de certification de niveau supérieur indépendantes. Il peut aussi y avoir moins de certificats que nécessaire, si on s'attend à ce que les receveurs aient des moyens de remplacement pour obtenir les certificats nécessaires (par exemple, à partir d'un ensemble précédent de certificats). Le certificat du signataire PEUT être inclus. L'utilisation de certificats d'attribut de version 1 est fortement déconseillée.

crls est une collection d'informations d'état de révocation. Il est destiné à ce que la collection contienne des informations

suffisantes pour déterminer si les certificats qui sont dans le champ Certificates sont valides, mais une telle correspondance n'est pas nécessaire. Les listes de révocation de certificats (CRL, *Certificate revocation List*) sont la principale source d'informations d'état de révocation. Il PEUT y avoir plus de CRL que nécessaire, et il PEUT aussi y en avoir moins que nécessaire.

signerInfos est une collection d'informations par signataire. Il PEUT y avoir un nombre quelconque d'éléments dans la collection, y compris zéro. Lorsque la collection représente plus d'une signature, la validation réussie d'une des signatures de la part d'un certain signataire devrait être traitée comme une signature réussie par ce signataire. Cependant, il y a certains environnements d'application où d'autres règles sont nécessaires. Les détails du type SignerInfo sont exposés au paragraphe 5.3. Comme chaque signataire peut employer une technique de signature numérique différente, et que de futures spécifications pourraient mettre à jour la syntaxe, toutes les mises en œuvre DOIVENT traiter avec bienveillance les versions non mises en œuvre de SignerInfo. De plus, comme toutes les mises en œuvre ne prendront pas en charge tous les algorithmes de signature possibles, toutes les mises en œuvre DOIVENT traiter avec bienveillance les algorithmes de signature non mis en œuvre lorsque il s'en rencontre.

5.2 Type EncapsulatedContentInfo

Le contenu est représenté dans le type EncapsulatedContentInfo :

```
EncapsulatedContentInfo ::= SEQUENCE {
    eContentType            ContentType,
    eContent [0]            CHAINE D'OCTET EXPLICITE OPTIONNELLE }

ContentType ::= IDENTIFIANT D'OBJET
```

Les champs de type EncapsulatedContentInfo ont la signification suivante :

eContentType est un identifiant d'objet. L'identifiant d'objet spécifie de façon univoque le type de contenu.

eContent est le contenu lui-même, porté comme une chaîne d'octets. Le eContent n'a pas besoin d'être codé en DER.

L'omission facultative du eContent au sein du champ EncapsulatedContentInfo rend possible la construction de "signatures externes". Dans le cas de signatures externes, le contenu à signer est absent de la valeur EncapsulatedContentInfo incluse dans le type de contenu signed-data. Si la valeur de eContent au sein de EncapsulatedContentInfo est absente, alors la signatureValue est calculée et le eContentType est alloué comme si la valeur de eContent était présente.

Dans le cas limite où il n'y aurait pas de signataire, la valeur EncapsulatedContentInfo "signée" ne serait pas pertinente. Dans ce cas, le type de contenu au sein de la valeur de EncapsulatedContentInfo étant "signée" DOIT être id-data (comme défini à la Section 4) et le champ de contenu de la valeur EncapsulatedContentInfo DOIT être omis.

5.2.1 Compatibilité avec PKCS n° 7

Ce paragraphe contient un mot d'avertissement pour les mises en œuvre qui souhaitent prendre en charge les types de contenu SignedData de la CMS et de PKCS n° 7 [RFC2315]. La CMS et PKCS n° 7 identifient tous deux le type de contenu encapsulé à un identifiant d'objet, mais le type ASN.1 du contenu lui-même est différent dans le type de contenu SignedData de PKCS n° 7.

PKCS n° 7 définit le contenu comme :

```
content [0] TOUT EXPLICITE DÉFINI PAR contentType OPTIONNEL
```

La CMS définit eContent comme :

```
eContent [0] CHAINE D'OCTET EXPLICITE OPTIONNELLE
```

La définition de la CMS est beaucoup plus facile à utiliser dans la plupart des applications, et elle est compatible à la fois avec S/MIME v2 et S/MIME v3. Les messages signés S/MIME qui utilisent la CMS et PKCS n° 7 sont compatibles parce que des formats de message signés identiques sont spécifiés dans la [RFC2311] pour S/MIME v2 , dans la [RFC2633] pour S/MIME v3, et dans la [RFC3851] pour S/MIME v3.1. S/MIME v2 encapsule le contenu MIME dans un type Data (c'est-à-dire, une CHAINE D'OCTET) portée dans le champ TOUT de contenu contentInfo SignedData, et S/MIME v3 porte le contenu MIME dans la CHAINE D'OCTET SignedData encapContentInfo eContent. Donc, dans S/MIME v2, S/MIME v3,

et S/MIME v3.1, le contenu MIME est placé dans une CHAINE D'OCTET et le résumé de message est calculé sur des portions identiques du contenu. C'est à dire, le résumé de message est calculé sur les octets qui comprennent la valeur de la CHAINE D'OCTET, ni l'étiquette ni les octets de longueur ne sont inclus.

Il y a des incompatibilités entre les types SignedData de CMS et de PKCS n° 7 lorsque le contenu encapsulé n'est pas formaté en utilisant le type Data. Par exemple, lorsque un reçu signé de la [RFC2634] est encapsulé dans le type SignedData de la CMS, la SEQUENCE Receipt est alors codée dans la CHAINE D'OCTETS eContent encapContentInfo de SignedData et le résumé de message est calculé en utilisant le codage entier de la SEQUENCE Receipt (y compris les octets étiquette, longueur et valeur). Cependant, si un Receipt signé de la RFC2634 est encapsulé dans le type SignedData de PKCS n° 7, la SEQUENCE Receipt est alors codée en DER [X.509-88] dans le champ TOUT du contenu de contentInfo de SignedData (une SEQUENCE, pas une CHAINE D'OCTETS). Donc, le résumé de message est calculé en utilisant seulement les octets de valeur du codage de la SEQUENCE Receipt.

On peut utiliser la stratégie suivante pour réaliser la rétro compatibilité avec PKCS n° 7 lors du traitement des types de contenu SignedData. Si la mise en œuvre n'est pas capable de décoder en ASN.1 le type SignedData en utilisant la syntaxe CMS de CHAINE D'OCTETS de eContent SignedData encapContentInfo, la mise en œuvre PEUT alors tenter de décoder le type SignedData en utilisant la syntaxe PKCS n° 7 TOUT du contentInfo SignedData et calculer le résumé de message en conséquence.

La stratégie suivante peut être utilisée pour réaliser la rétro compatibilité avec PKCS n° 7 lors de la création d'un type de contenu SignedData dans lequel le contenu encapsulé n'est pas formaté en utilisant le type Data. Les mises en œuvre PEUVENT examiner la valeur du eContentType, et ajuster ensuite le codage DER attendu du eContent sur la base de la valeur de l'identifiant d'objet. Par exemple, pour prendre en charge Authenticode de Microsoft [MSAC], les informations suivantes PEUVENT être incluses :

L'identifiant d'objet eContentType est réglé à { 1 3 6 1 4 1 311 2 1 4 }

eContent contient les informations de signature Authenticode codées en DER

5.3 Type SignerInfo

Les informations sur le signataire sont représentées par le type SignerInfo :

```
SignerInfo ::= SEQUENCE {
    version                 CMSVersion,
    sid                     SignerIdentifier,
    digestAlgorithm         DigestAlgorithmIdentifier,
    signedAttrs [0]         IMPLICITE SignedAttributes OPTIONNEL,
    signatureAlgorithm      SignatureAlgorithmIdentifier,
    signature               SignatureValue,
    unsignedAttrs [1]       IMPLICITE UnsignedAttributes OPTIONNEL }

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

SignedAttributes ::= REGLER LA TAILLE DE (1..MAX) DE Attribute

UnsignedAttributes ::= REGLER LA TAILLE DE (1..MAX) DE Attribute

Attribute ::= SEQUENCE {
    attrType                IDENTIFIANT D'OBJET,
    attrValues              ENSEMBLE DE AttributeValue }

AttributeValue ::= TOUT

SignatureValue ::= CHAINE D'OCTETS
```

Les champs de type SignerInfo ont la signification suivante :

version est le numéro de version de la syntaxe. Si le SignerIdentifier est le CHOIX issuerAndSerialNumber, la version

DOIT alors être 1. Si le SignerIdentifier est subjectKeyIdentifier, la version DOIT alors être 3.

sid spécifie le certificat du signataire (et par là même la clé publique du signataire). Le receveur a besoin de la clé publique du signataire pour vérifier la signature. SignerIdentifier donne deux solutions alternatives pour spécifier la clé publique du signataire. La solution issuerAndSerialNumber identifie le certificat du signataire par le nom distinctif du producteur et le numéro de série du certificat ; le subjectKeyIdentifier identifie le certificat du signataire par un identifiant de clé. Lorsque il est fait référence à un certificat X.509, l'identifiant de clé correspond à la valeur de l'extension X.509 subjectKeyIdentifier. Lorsque il est fait référence à d'autres formats de certificat, les documents qui spécifient le format du certificat et leur utilisation avec la CMS doivent inclure des détails sur la mise en correspondance de l'identifiant de clé avec le champ approprié du certificat. Les mises en œuvre DOIVENT prendre en charge la réception des formes issuerAndSerialNumber et subjectKeyIdentifier de SignerIdentifier. Lorsque elles génèrent un SignerIdentifier, les mises en œuvre PEUVENT prendre en charge une des formes (soit issuerAndSerialNumber, soit subjectKeyIdentifier) et toujours l'utiliser, ou bien les mises en œuvre PEUVENT mélanger arbitrairement les deux formes. Cependant, subjectKeyIdentifier DOIT être utilisé pour se référer à une clé publique contenue dans un certificat non X.509.

digestAlgorithm identifie l'algorithme de résumé de message, et tous les paramètres associés, utilisés par le signataire. Le résumé de message est calculé soit sur le contenu qui est signé, soit sur le contenu et les attributs signés en utilisant le processus décrit au paragraphe 5.4. L'algorithme de résumé de message DEVRAIT être de ceux énumérés dans le champ digestAlgorithms des SignedData associées. Les mises en œuvre PEUVENT échouer à valider les signatures qui utilisent un algorithme de résumé qui n'est pas inclus dans l'ensemble digestAlgorithm des SignedData.

signedAttrs est une collection d'attributs qui sont signés. Le champ est facultatif, mais il DOIT être présent si le type de contenu de la valeur EncapsulatedContentInfo qui est signée n'est pas id-data. SignedAttributes DOIT être codé en DER, même si le reste de la structure est codé en BER. Des types d'attribut utiles, tels que l'heure de signature, sont définis à la Section 11. Si le champ est présent, il DOIT contenir, au minimum, les deux attributs suivants :

- un attribut Type de contenu qui a comme valeur le type de contenu de la valeur EncapsulatedContentInfo qui est signée. Le paragraphe 11.1 définit l'attribut Type de contenu. Cependant, l'attribut type de contenu NE DOIT PAS être utilisé au titre d'un attribut non signé Contreseing comme défini au paragraphe 11.4.

- un attribut Résumé de message qui a comme valeur le résumé de message du contenu. Le paragraphe 11.2 définit l'attribut Résumé de message.

signatureAlgorithm identifie l'algorithme de signature, et tous les paramètres associés, utilisés par le signataire pour générer la signature numérique.

signature est le résultat de la génération de signature numérique, en utilisant le résumé de message et la clé privé du signataire. Les détails de la signature dépendent de l'algorithme de signature employé.

unsignedAttrs est une collection d'attributs qui ne sont pas signés. Le champ est facultatif. Des types d'attribut utiles, tels que les Contreseings, sont définis à la Section 11.

Les champs de type SignedAttribute et UnsignedAttribute ont la signification suivante :

attrType indique le type d'attribut. C'est un identifiant d'objet.

attrValues est un ensemble de valeurs qui comprennent l'attribut. Le type de chaque valeur dans l'ensemble peut être déterminé de façon univoque par attrType. Le attrType peut imposer des restrictions au nombre d'éléments dans l'ensemble.

5.4 Processus de calcul du résumé de message

Le processus de calcul du résumé de message calcule un résumé de message soit sur le contenu à signer, soit sur le contenu avec les attributs signés. Dans l'un et l'autre cas, l'entrée initiale du processus de calcul du résumé de message est la "valeur" du contenu encapsulé à signer. Précisément, l'entrée initiale est la CHAINE D'OCTETS encapContentInfo eContent à laquelle le processus de signature est appliqué. Seuls les octets comportant la valeur de la CHAINE D'OCTETS eContent sont en entrée dans l'algorithme de résumé de message, et non les octets d'étiquette ou de longueur.

Le résultat du processus de calcul du résumé de message dépend de la présence du champ signedAttrs. Lorsque le champ est absent, le résultat est juste le résumé de message du contenu comme décrit ci-dessus. Cependant, lorsque le champ est présent, le résultat est le résumé de message du codage DER complet de la valeur SignedAttrs contenue dans le champ signedAttrs. Comme la valeur SignedAttrs, lorsque elle est présente, doit contenir les attributs Type de contenu et Résumé

de message, ces valeurs sont indirectement incluses dans le résultat. L'attribut Type de contenu NE DOIT PAS être inclus dans un attribut Contreseing non signé, comme défini au paragraphe 11.4. Un codage séparé du champ signedAttrs est effectué pour le calcul du résumé de message. L'étiquette IMPLICITE [0] dans le signedAttrs n'est pas utilisée pour le codage en DER, on utilise plutôt une étiquette ENSEMBLE EXPLICITE DE. C'est-à-dire que le codage en DER de l'étiquette ENSEMBLE EXPLICITE DE DOIT être inclus dans le calcul du résumé de message avec les octets de longueur et de contenu de la valeur SignedAttributes, plutôt que l'étiquette IMPLICITE [0],.

Lorsque le champ signedAttrs est absent, seuls les octets comportant la valeur de la CHAINE D'OCTETS SignedData encapContentInfo eContent (par exemple, le contenu d'un fichier) sont en entrée pour le calcul du résumé de message. Cela présente l'avantage que la longueur du contenu signé n'a pas besoin d'être connue à l'avance du processus de génération de la signature.

Bien que les octets de l'étiquette CHAINE D'OCTETS du eContent encapContentInfo et de longueur ne soient pas inclus dans le calcul du résumé de message, ils sont protégés par d'autres moyens. Les octets de longueur sont protégés par la nature de l'algorithme de résumé de message car il est irréalisable de trouver par le calcul deux contenus de message distincts de quelque longueur que ce soit qui aient le même résumé de message.

5.5 Processus de génération de signature

L'entrée du processus de génération de signature inclut le résultat du processus de calcul du résumé de message et la clé privée du signataire. Les détails de la génération de signature dépendent de l'algorithme de signature employé. L'identifiant d'objet, ainsi que tous les paramètres qui spécifient l'algorithme de signature employé par le signataire sont portés dans le champ signatureAlgorithm. La valeur de la signature générée par le signataire DOIT être codée comme une CHAINE D'OCTETS et porter le champ Signature.

5.6 Processus de vérification de signature

L'entrée du processus de vérification de signature inclut le résultat du processus de calcul du résumé de message et la clé publique du signataire. Le receveur PEUT obtenir la clé publique correcte pour le signataire par tous moyens, mais la méthode préférée est à partir d'un certificat obtenu du champ SignedData des certificats. Le choix et la validation de la clé publique du signataire PEUVENT se fonder sur la validation du chemin de certification (voir la [RFC5280]) ainsi que sur d'autres contextes externes, mais cela sort du domaine d'application du présent document. Les détails de la vérification de signature dépendent de l'algorithme de signature employé.

Le receveur NE DOIT PAS s'appuyer sur des valeurs de résumé de message calculées par l'origine du message. Si les signerInfo des SignedData comportent des signedAttributes, le résumé de message du contenu DOIT être calculé comme décrit au paragraphe 5.4. Pour que la signature soit valide, la valeur du résumé de message calculée par le receveur DOIT être la même que la valeur de l'attribut messageDigest inclus dans le signedAttributes des signerInfo des SignedData.

Si les signerInfo des SignedData incluent des signedAttributes, la valeur de l'attribut type de contenu DOIT alors correspondre à la valeur du eContentType des encapContentInfo des SignedData.

6. Type de contenu Enveloped-data

Le type de contenu enveloped-data consiste en un contenu chiffré de tout type et des clés de chiffrement de contenu chiffrées pour un ou plusieurs receveurs. La combinaison du contenu chiffré et d'une clé de chiffrement de contenu chiffrée pour un receveur est une "enveloppe numérique" pour ce receveur. Tout type de contenu peut être enveloppé pour un nombre arbitraire de receveurs en utilisant une des techniques de gestion de clé prises en charge pour chaque receveur.

L'application normale du type de contenu enveloped-data va représenter une ou plusieurs enveloppes numériques des receveurs sur les types de contenu des données ou de contenu des données signées.

Enveloped-data est construit selon les étapes suivantes :

1. Une clé de chiffrement de contenu est générée au hasard pour un algorithme particulier de chiffrement de contenu.

2. La clé de chiffrement de contenu est chiffrée pour chaque receveur. Les détails de ce chiffrement dépendent de l'algorithme de gestion de clé utilisé, mais quatre techniques générales sont acceptées :

- transport de clé : la clé de chiffrement de contenu est chiffrée avec la clé publique du receveur ;
- accord de clé : la clé publique du receveur et la clé privée de l'envoyeur sont utilisées pour générer une paire de clés symétriques couplées, puis la clé de chiffrement de contenu est chiffrée avec la clé symétrique couplée ;
- clés symétriques de chiffrement de clés : la clé de chiffrement de contenu est chiffrée avec une clé symétrique de chiffrement de clé distribuée précédemment ;
- des mots de passe : la clé de chiffrement de contenu est chiffrée avec une clé de chiffrement de clé qui est déduite d'un mot de passe ou autre valeur de secret partagé.

3. Pour chaque receveur, la clé de chiffrement de contenu chiffrée et les autres informations spécifiques du receveur sont collectées dans une valeur RecipientInfo, définie au paragraphe 6.2.

4. Le contenu est chiffré avec la clé de chiffrement de contenu. Le chiffrement du contenu peut exiger que le contenu soit bourré jusqu'à former un multiple d'une certaine taille de bloc ; voir au paragraphe 6.3.

5. Les valeurs RecipientInfo pour tous les receveurs sont collectées avec le contenu chiffré pour former une valeur EnvelopedData comme défini au paragraphe 6.1.

Un receveur ouvre l'enveloppe numérique en déchiffrant une des clés de chiffrement de contenu chiffrée et en déchiffrant le contenu chiffré avec la clé de chiffrement de contenu récupérée.

La présente section est divisée en quatre parties. La première décrit le type EnvelopedData de niveau supérieur, la seconde décrit le type d'informations par receveur RecipientInfo, et les troisième et quatrième parties décrivent les processus de chiffrement de contenu et de chiffrement de clé.

6.1 Type EnvelopedData

L'identifiant d'objet suivant identifie le type de contenu enveloped-data :

```
IDENTIFIANT D'OBJET id-envelopedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7)
  3 }
```

Le type de contenu enveloped-data devra avoir le type ASN.1 EnvelopedData :

```
EnvelopedData ::= SEQUENCE {
    version                 CMSVersion,
    originatorInfo [0]      IMPLICITE OriginatorInfo OPTIONNEL,
    recipientInfos          RecipientInfos,
    encryptedContentInfo    EncryptedContentInfo,
    unprotectedAttrs [1]    IMPLICITE UnprotectedAttributes OPTIONNEL }

OriginatorInfo ::= SEQUENCE {
    certs [0]               IMPLICITE CertificateSet OPTIONNEL,
    crls [1]                IMPLICITE RevocationInfoChoices OPTIONNEL }

RecipientInfos ::= REGLE TAILLE (1..MAX) DE RecipientInfo

EncryptedContentInfo ::= SEQUENCE {
    contentType             ContentType,
    contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
    encryptedContent [0]    IMPLICITE EncryptedContent OPTIONNEL }

EncryptedContent ::= CHAINE D'OCTETS

UnprotectedAttributes ::= REGLE TAILLE (1..MAX) DE Attribute
```

Les champs de type EnvelopedData ont les significations suivantes :

version est le numéro de version de la syntaxe. La valeur appropriée dépend de originatorInfo, RecipientInfo, et unprotectedAttrs. La version DOIT être allouée comme suit :

```
SI (originatorInfo est présent) ET
    ((des certificats avec un type de autre sont présents) OU
    (des crl avec un type de autre sont présents))
ALORS version est 4
AUTREMENT
    SI ((originatorInfo est présent) ET
        (des certificats d'attribut de version 2 sont présents)) OU
        (des structures RecipientInfo incluent pwri) OU
        (des structures RecipientInfo incluent ori)
    ALORS version est 3
    AUTREMENT
        SI (originatorInfo est absent) ET
            (unprotectedAttrs est absent) ET
            (toutes les structures RecipientInfo sont de version 0)
        ALORS version est 0
        AUTREMENT version est 2
```

originatorInfo donne facultativement des informations sur l'origine. Il n'est présent que si c'est exigé par l'algorithme de gestion de clé. Il peut contenir des certificats et des CRL :

certs est une collection de certificats. certs peut contenir des certificats d'origine associés à plusieurs différents algorithmes de gestion de clé. certs peut aussi contenir des certificats d'attribut associés à l'origine. Les certificats contenus dans les certs sont destinés à être suffisants pour que tous les receveurs construisent des chemins de certification à artir d'une "racine" ou "autorité de certification de niveau supérieur" reconnue. Cependant, certs peut contenir plus de certificats que nécessaire, et il peut y avoir suffisamment de certificats pour faire des chemins de certification à partir de deux autorités de certification de niveau supérieur indépendantes ou plus. Autrement, certs peut contenir moins de certificats que nécessaire, si on s'attend à ce que les receveurs aient d'autres moyens d'obtenir les certificats nécessaires (par exemple, à partir d'un ensemble de certificats précédent).

crls est une collection de CRL. Il est destiné à ce que l'ensemble contienne des informations suffisantes pour déterminer si les certificats dans le champ certs sont ou non valides, mais une telle correspondance n'est pas nécessaire. Il PEUT y avoir plus de CRL que nécessaire, et il PEUT aussi y avoir moins de CRL que nécessaire.

recipientInfos est une collection d'informations par receveur. Il DOIT y avoir au moins un élément dans la collection.

encryptedContentInfo sont les informations de contenu chiffrées.

unprotectedAttrs est une collection d'attributs qui ne sont pas chiffrés. Le champ est facultatif. Les types d'attribut utiles sont définis à la Section 11.

Les champs de type EncryptedContentInfo ont les significations suivantes :

contentType indique le type de contenu.

contentEncryptionAlgorithm identifie l'algorithme de chiffrement de contenu, et tous les paramètres associés, utilisé pour chiffrer le contenu. Le processus de chiffrement du contenu est décrit au paragraphe 6.3. Le même algorithme de chiffrement de contenu et la même clé de chiffrement de contenu sont utilisés pour tous les receveurs.

encryptedContent est le résultat du chiffrement du contenu. Le champ est facultatif, et si le champ n'est pas présent, la valeur qu'on entend lui donner doit être fournie par d'autres moyens.

Le champ recipientInfos vient avant le champ encryptedContentInfo de sorte que la valeur EnvelopedData peut être traitée en un seul passage.

6.2 Type RecipientInfo

Les informations par receveur sont représentées dans le type RecipientInfo. RecipientInfo a un format différent pour chacune des techniques de gestion de clé prise en charge. Toute technique de gestion de clé peut être utilisée pour chaque receveur du même contenu chiffré. Dans tous les cas, la clé de chiffrement de contenu chiffrée est transférée à un ou plusieurs receveurs.

Comme toutes les mises en œuvre ne vont pas prendre en charge tous les algorithmes de gestion de clé possibles, toutes les mises en œuvre DOIVENT traiter avec bienveillance les algorithmes non mis en œuvre lorsque il s'en rencontre. Par exemple, si un receveur reçoit une clé de chiffrement de contenu chiffrée dans sa clé publique RSA qui utilise RSA-OAEP (*Optimal Asymmetric Encryption Padding*, bourrage optimal de chiffrement asymétrique) et si la mise en œuvre ne prend en charge que RSA PKCS n° 1 v1.5, un échec en douceur doit être mis en œuvre.

Les mises en œuvre DOIVENT accepter le transport de clé, l'accord de clé, et les clés de chiffrement de clé symétrique à distribution préalable, comme représenté respectivement par ktri, kari, et kekri. Les mises en œuvre PEUVENT prendre en charge la gestion de clé fondée sur le mot de passe, comme représenté par pwri. Les mises en œuvre PEUVENT prendre en charge toute autre technique de gestion de clé, comme représenté par ori. Comme chaque receveur peut employer une technique de gestion de clé différente et que de futures spécifications pourraient définir des techniques de gestion de clé supplémentaires, toutes les mises en œuvre DOIVENT traiter avec bienveillance les solutions de remplacement non mises en œuvre au sein du CHOIX RecipientInfo, toutes les mises en œuvre DOIVENT traiter avec bienveillance les versions non mises en œuvre de solutions de remplacement par ailleurs prises en charge au sein de CHOIX RecipientInfo, et toutes les mises en œuvre DOIVENT traiter avec bienveillance les ori de remplacement non mises en œuvre ou inconnues.

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

EncryptedKey ::= CHAINE D'OCTETS
```

6.2.1 Type KeyTransRecipientInfo

Les informations par receveur en utilisant le transport de clés sont représentées dans le type KeyTransRecipientInfo. Chaque instance de KeyTransRecipientInfo transfère la clé de chiffrement de contenu à un receveur.

```
KeyTransRecipientInfo ::= SEQUENCE {
    version                 CMSVersion,                     - toujours réglé à 0 ou 2
    rid                     RecipientIdentifier,
    keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
    encryptedKey            EncryptedKey }

RecipientIdentifier ::= CHOIX {
    issuerAndSerialNumber   IssuerAndSerialNumber,
    subjectKeyIdentifier [0]  SubjectKeyIdentifier }
```

Les champs du type KeyTransRecipientInfo ont les significations suivantes :

version est le numéro de version de la syntaxe. Si le RecipientIdentifier est le CHOIX issuerAndSerialNumber, la version DOIT alors être 0. Si le RecipientIdentifier est subjectKeyIdentifier, la version DOIT alors être 2.

rid spécifie le certificat ou la clé du receveur qui a été utilisée par l'envoyeur pour protéger la clé de chiffrement de contenu. La clé de chiffrement de contenu est chiffrée avec la clé publique du receveur. Le RecipientIdentifier fournit deux solutions de remplacement pour spécifier le certificat du receveur, et donc la clé publique du receveur. Le certificat du receveur doit contenir une clé publique de transport de clé. Donc, un certificat X.509 version 3 de receveur qui contient une extension d'usage de clé DOIT affirmer le bit keyEncipherment. La solution issuerAndSerialNumber identifie le certificat du receveur par le nom distinctif du producteur et le numéro de série du certificat ; le subjectKeyIdentifier identifie le certificat du receveur par un identifiant de clé. Lorsque un certificat X.509 est référencé, l'identifiant de clé correspond à la valeur d'extension subjectKeyIdentifier X.509. Lorsque d'autres formats de certificat sont référencés, les documents qui spécifient le format du certificat et son utilisation avec la CMS doivent inclure des détails sur la correspondance entre l'identifiant de clé et le champ de certificat approprié. Pour le traitement du receveur, les mises en œuvre DOIVENT accepter ces deux solutions de remplacement pour spécifier le certificat du receveur. Pour le traitement de l'envoyeur, les mises en œuvre DOIVENT accepter au moins une de ces solutions.

keyEncryptionAlgorithm identifie l'algorithme de chiffrement de clé, et tous les paramètres associés, utilisés pour chiffrer la clé de chiffrement de contenu pour le receveur. Le processus de chiffrement de clé est décrit au paragraphe 6.4.

encryptedKey est le résultat du chiffrement de la clé de chiffrement de contenu pour le receveur.

6.2.2 Type KeyAgreeRecipientInfo

Les informations sur le receveur en utilisant l'accord de clé sont représentées dans le type KeyAgreeRecipientInfo. Chaque instance de KeyAgreeRecipientInfo va transférer la clé de chiffrement de contenu à un ou plusieurs receveurs qui utilisent le même algorithme d'accord de clé et paramètres de domaine pour cet algorithme.

```
KeyAgreeRecipientInfo ::= SEQUENCE {
    version                 CMSVersion,                          -- toujours réglé à 3
    originator [0]          EXPLICITE OriginatorIdentifierOrKey,
    ukm [1]                 EXPLICITE UserKeyingMaterial OPTIONNEL,
    keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
    recipientEncryptedKeys  RecipientEncryptedKeys }

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

OriginatorPublicKey ::= SEQUENCE {
    algorithm               AlgorithmIdentifier,
    publicKey               CHAINE BINAIRE }

RecipientEncryptedKeys ::= SEQUENCE DE RecipientEncryptedKey

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

KeyAgreeRecipientIdentifier ::= CHOIX {
    issuerAndSerialNumber   IssuerAndSerialNumber,
    rKeyId [0] IMPLICIT     RecipientKeyIdentifier }

RecipientKeyIdentifier ::= SEQUENCE {
    subjectKeyIdentifier    SubjectKeyIdentifier,
    date GeneralizedTime    OPTIONNEL,
    other OtherKeyAttribute OPTIONNEL }

SubjectKeyIdentifier ::= CHAINE D'OCTETS
```

Les champs du type KeyAgreeRecipientInfo ont les significations suivantes :

version est le numéro de version de la syntaxe. Il DOIT toujours être 3.

originator est un CHOIX avec trois alternatives spécifiant la clé publique d'accord de clé de l'envoyeur. L'envoyeur utilise la clé privée correspondante et la clé publique du receveur pour générer une paire de clés. La clé de chiffrement de contenu est chiffrée avec la clé de la paire. La solution de remplacement issuerAndSerialNumber identifie le certificat de l'envoyeur, et par là, la clé publique de l'envoyeur, par le nom distinctif du producteur et le numéro de série du certificat. La solution subjectKeyIdentifier identifie le certificat de l'envoyeur, et par là, la clé publique de l'envoyeur, par un identifiant de clé. Lorsque un certificat X.509 est référencé, l'identifiant de clé correspond à la valeur d'extension X.509 subjectKeyIdentifier. Lorsque d'autres formats de certificat sont référencés, les documents qui spécifient le format du certificat et son utilisation avec la CMS doivent inclure des détails sur la mise en correspondance de l'identifiant de clé avec le champ de certificat approprié. La solution originatorKey inclut l'identifiant d'algorithme et la clé publique d'accord de clé de l'envoyeur. Cette solution de remplacement permet l'anonymat de l'origine car la clé publique n'est pas certifiée. Les mises en œuvre DOIVENT accepter les trois solutions de remplacement pour spécifier la clé publique de l'envoyeur.

ukm est facultatif. Avec certains algorithmes d'accord de clé, l'envoyeur fournit un matériel de clés d'utilisateur (UKM, *User Keying Material*) pour assurer qu'une clé différente est générée chaque fois que les deux mêmes parties génèrent une paire de clés. Les mises en œuvre DOIVENT accepter une SÉQUENCE KeyAgreeRecipientInfo qui comporte un champ ukm. Les mises en œuvre qui n'acceptent pas les algorithmes d'accord de clé qui utilisent des UKM DOIVENT traiter avec bienveillance la présence d'UKM.

keyEncryptionAlgorithm identifie l'algorithme de chiffrement de clé, et tous les paramètres associés, utilisés pour chiffrer la clé de chiffrement de contenu avec la clé de chiffrement de clé. Le processus de chiffrement de clé est décrit au paragraphe 6.4.

recipientEncryptedKeys comporte un identifiant de receveur et une clé chiffrée pour un ou plusieurs receveurs. Le KeyAgreeRecipientIdentifier est un CHOIX avec deux solutions pour spécifier le certificat du receveur, et par là, la clé publique du receveur, qui a été utilisée par l'envoyeur pour générer une paire de clés de chiffrement de clé. Le certificat du receveur doit contenir une clé publique d'accord de clé. Donc, un certificat X.509 version 3 de receveur qui contient une extension d'usage de clé DOIT établir le bit keyAgreement. La clé de chiffrement de contenu est chiffrée avec la paire de clés de chiffrement de clé. La solution issuerAndSerialNumber identifie le certificat du receveur par le nom distinctif du producteur et le numéro de série du certificat ; le RecipientKeyIdentifier est décrit ci-dessous. La encryptedKey est le résultat du chiffrement de la clé de chiffrement de contenu avec la paire de clés de chiffrement de clé générée en utilisant l'algorithme d'accord de clé. Les mises en œuvre DOIVENT accepter les deux solutions pour spécifier le certificat du receveur.

Les champs de type RecipientKeyIdentifier ont les significations suivantes :

subjectKeyIdentifier identifie le certificat du receveur par un identifiant de clé. Lorsque un certificat X.509 est référencé, l'identifiant de clé correspond à la valeur d'extension X.509 subjectKeyIdentifier. Lorsque d'autres formats de certificat sont référencés, les documents qui spécifient le format du certificat et leur utilisation par la CMS doivent inclure les détails sur la mise en correspondance de l'identifiant de clé avec le champ de certificat approprié.

La date est facultative. Lorsque présente, la date spécifie quel UKM préalablement distribué du receveur a été utilisé par l'envoyeur.

other est facultatif. Lorsque il est présent, ce champ contient des informations supplémentaires utilisées par le receveur pour localiser le matériel de clé publique utilisé par l'envoyeur.

6.2.3 Type KEKRecipientInfo

Les informations de receveur qui utilisent des clés symétriques distribuées au préalable sont représentées dans le type KEKRecipientInfo. Chaque instance de KEKRecipientInfo va transférer la clé de chiffrement de contenu à un ou plusieurs receveurs qui ont la clé de chiffrement de clé distribuée au préalable.

```
KEKRecipientInfo ::= SEQUENCE {
    version                 CMSVersion,                          -- toujours réglé à 4
    kekid                   KEKIdentifier,
    keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
    encryptedKey            EncryptedKey }

KEKIdentifier ::= SEQUENCE {
    keyIdentifier           CHAINE D'OCTETS,
    date                    GeneralizedTime OPTIONNEL,
    other                   OtherKeyAttribute OPTIONNEL }
```

Les champs de type KEKRecipientInfo ont les significations suivantes :

version est le numéro de version de la syntaxe. Il DOIT toujours être 4.

kekid spécifie une clé de chiffrement de clé symétrique distribuée au préalable à l'envoyeur et à un ou plusieurs receveurs.

keyEncryptionAlgorithm identifie l'algorithme de chiffrement de clé, et tous paramètres associés, utilisé pour chiffrer la clé de chiffrement de contenu avec la clé de chiffrement de clé. Le processus de chiffrement de clé est décrit au paragraphe 6.4.

encryptedKey est le résultat du chiffrement de la clé de chiffrement de contenu avec la clé de chiffrement de clé.

Les champs du type KEKIdentifier ont les significations suivantes :

keyIdentifier identifie la clé de chiffrement de clé qui a été distribuée au préalable à l'envoyeur et à un ou plusieurs receveurs.

date est facultatif. Lorsque présente, la date spécifie une seule clé de chiffrement de clé parmi un ensemble distribué au préalable.

other est facultatif. Lorsque présent, ce champ contient des informations supplémentaires utilisées par le receveur pour déterminer la clé de chiffrement de clé utilisée par l'envoyeur.

6.2.4 Type PasswordRecipientInfo

Les informations de receveur qui utilisent un mot de passe ou une valeur de secret partagé sont représentées dans le type PasswordRecipientInfo. Chaque instance de PasswordRecipientInfo va transférer la clé de chiffrement de contenu à un ou plusieurs receveurs qui possèdent le mot de passe ou la valeur du secret partagé.

Le type PasswordRecipientInfo est spécifié dans la [RFC3211]. La structure PasswordRecipientInfo est répétée ici par souci de complétude.

```
PasswordRecipientInfo ::= SEQUENCE {
    version                   CMSVersion,                                  -- Toujours réglé à 0
    keyDerivationAlgorithm [0] KeyDerivationAlgorithmIdentifier OPTIONNEL,
    keyEncryptionAlgorithm    KeyEncryptionAlgorithmIdentifier,
    encryptedKey              EncryptedKey }
```

Les champs de type PasswordRecipientInfo ont les significations suivantes :

version est le numéro de version de la syntaxe. Il DOIT toujours être 0.

keyDerivationAlgorithm identifie l'algorithme de déduction de clé, et tous les paramètres associés, utilisés pour déduire la clé de chiffrement de clé du mot de passe ou de la valeur de secret partagé. Si ce champ est absent, la clé de chiffrement de clé est fournie à partir d'une source externe, par exemple un jeton de chiffrement matériel comme une carte à puce.

keyEncryptionAlgorithm identifie l'algorithme de chiffrement, et tous paramètres associés, utilisé pour chiffrer la clé de chiffrement de contenu avec la clé de chiffrement de clé.

encryptedKey est le résultat du chiffrement de la clé de chiffrement de contenu avec la clé de chiffrement de clé.

6.2.5 Type OtherRecipientInfo

Les informations sur le receveur pour les techniques de gestion de clé supplémentaires sont représentées dans le type OtherRecipientInfo. Le type OtherRecipientInfo permettra de spécifier dans de futurs documents d'autres techniques de gestion de clé que le transport de clé, l'accord de clé, la clé de chiffrement de clés symétriques à distribution préalable, et la gestion de clé fondée sur le mot de passe. Un identifiant d'objet identifie de façon univoque de telles techniques de gestion de clé.

```
OtherRecipientInfo ::= SEQUENCE {
    oriType                   IDENTIFIANT D'OBJET,
    oriValue                  TOUT DÉFINI PAR oriType }
```

Les champs de type OtherRecipientInfo ont les significations suivantes :

oriType identifie la technique de gestion de clé.

oriValue contient les éléments de données de protocole nécessaires pour un receveur qui utilise la technique de gestion de clé identifiée.

6.3 Processus de chiffrement de contenu

La clé de chiffrement de contenu pour l'algorithme de chiffrement de contenu désiré est générée de façon aléatoire. Les données à protéger sont bourrées comme décrit ci-dessous, puis les données avec leur bourrage sont chiffrées en utilisant la clé de chiffrement de contenu. L'opération de chiffrement transpose une chaîne arbitraire d'octets (les données) en une autre chaîne d'octets (le texte chiffré) sous le contrôle d'une clé de chiffrement de contenu. Les données chiffrées sont incluses dans la CHAINE D'OCTETS encryptedContent encryptedContentInfo de EnvelopedData .

Certains algorithmes de chiffrement de contenu supposent que la longueur des entrées est un multiple de k octets, où k est supérieur à un. Pour de tels algorithmes, l'entrée devra être bourrée en fin avec les k-(l^e mod k) octets ayant tous la valeur k-(l^e mod k) où l^e est la longueur de l'entrée. Autrement dit, l'entrée est bourrée en fin avec une des chaînes suivantes :

01 -- si l^e mod k = k-1
02 02 – si l^e mod k = k-2
.
.
.
k k ... k k -- si l^e mod k = 0

Le bourrage peut être retiré sans ambiguïté car toutes les entrées sont bourrées, y compris les valeurs d'entrée qui sont déjà un multiple de la taille de bloc, et aucune chaîne de bourrage n'est un suffixe d'une autre. Cette méthode de bourrage est bien définie si et seulement si k est inférieur à 256.

6.4 Processus de chiffrement de clés

L'entrée du processus de chiffrement de clé -- la valeur fournie à l'algorithme de chiffrement de clé du receveur -- est juste la "valeur" de la clé de chiffrement de contenu.

Toutes les techniques de gestion de clé susmentionnées peuvent être utilisées pour chaque receveur du même contenu chiffré.

7. Type de contenu Digested-data

Le type de contenu digested-data consiste en un contenu de n'importe quel type et d'un résumé de message du contenu.

Normalement, le type de contenu digested-data est utilisé pour assurer l'intégrité du contenu, et le résultat devient généralement une entrée du type de contenu enveloped-data .

Les étapes suivantes sont suivies pour la construction des digested-data :

1. Un résumé de message est calculé sur le contenu avec un algorithme de résumé de message.

2. L'algorithme de résumé de message et le résumé de message sont collectés avec le contenu dans une valeur DigestedData.

Un receveur vérifie le résumé de message en comparant le résumé de message à un résumé de message calculé de façon indépendante.

L'identifiant d'objet suivant identifie le type de contenu digested-data :

IDENTIFIANT D'OBJET id-digestedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 5 }

Le type de contenu digested-data devra avoir le type ASN.1 DigestedData :

```
DigestedData ::= SEQUENCE {
    version              CMSVersion,
    digestAlgorithm      DigestAlgorithmIdentifier,
    encapContentInfo     EncapsulatedContentInfo,
    digest Digest }

Digest ::= CHAINE D'OCTETS
```

Les champs de type DigestedData ont les significations suivantes :

version est le numéro de version de la syntaxe. Si le type de contenu encapsulé est id-data, la valeur de la version DOIT alors être 0 ; cependant, si le type de contenu encapsulé est autre que id-data, la valeur de la version DOIT alors être 2.

digestAlgorithm identifie l'algorithme de résumé de message, et tous paramètres associés, avec lequel le contenu est résumé. Le processus de résumé de message est le même qu'au paragraphe 5.4 dans le cas où il n'y a pas d'attribut signé.

encapContentInfo est le contenu qui est résumé, comme défini au paragraphe 5.2.

digest est le résultat du processus de résumé de message.

L'ordre des champs digestAlgorithm, encapContentInfo, et digest rend possible de traiter une valeur DigestedData en une seule fois.

8. Type de contenu Encrypted-data

Le type de contenu encrypted-data consiste en un contenu chiffré de tout type. À la différence du type de contenu enveloped-data, le type de contenu encrypted-data n'a de clé de chiffrement de contenu ni chiffrée ni de receveur. Les clés DOIVENT être gérées par d'autres moyens.

L'application normale du type de contenu encrypted-data va être de chiffrer le contenu du type de contenu data pour une mémorisation locale, peut-être lorsque la clé de chiffrement est déduite d'un mot de passe.

L'identifiant d'objet suivant identifie le type de contenu encrypted-data :

IDENTIFIANT D'OBJETid-encryptedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 6 }

Le type de contenu encrypted-data devra avoir le type ASN.1 EncryptedData :

```
EncryptedData ::= SEQUENCE {
    version                 CMSVersion,
    encryptedContentInfo    EncryptedContentInfo,
    unprotectedAttrs [1]    IMPLICITE UnprotectedAttributes OPTIONNEL }
```

Les champs de type EncryptedData ont les significations suivantes :

version est le numéro de version de la syntaxe. Si unprotectedAttrs est présent, la version DOIT alors être 2. Si unprotectedAttrs est absent, la version DOIT alors être 0.

encryptedContentInfo sont les informations de contenu chiffrées, comme défini au paragraphe 6.1.

unprotectedAttrs est une collection d'attributs qui ne sont pas chiffrés. Le champ est facultatif. Les types d'attribut utiles sont définis à la Section 11.

9. Type de contenu Authenticated-data

Le type de contenu authenticated-data consiste en un contenu de tout type, d'un code d'authentification de message (MAC), et de clés d'authentification chiffrées pour un ou plusieurs receveurs. La combinaison du MAC et d'une clé d'authentification chiffrée pour un receveur est nécessaire pour que le receveur vérifie l'intégrité du contenu. Tout type de contenu peut être protégé en intégrité pour un nombre quelconque de receveurs.

Le processus par lequel sont construites les données authentifiées implique les étapes suivantes :

1. Une clé d'authentification de message pour un algorithme particulier d'authentification de message est généré de façon aléatoire.
2. La clé d'authentification de message est chiffrée pour chaque receveur. Les détails de ce chiffrement dépendent de l'algorithme de gestion de clé utilisé.
3. Pour chaque receveur, la clé chiffrée d'authentification de message et les autres informations spécifiques du receveur sont collectées dans une valeur RecipientInfo, définie au paragraphe 6.2.
4. En utilisant la clé d'authentification de message, l'origine calcule une valeur de MAC sur le contenu. Si l'origine authentifie des informations en plus du contenu (voir au paragraphe 9.2) un résumé de message est calculé sur le contenu, le résumé de message du contenu et les autres informations sont authentifiés en utilisant la clé d'authentification de message, et le résultat devient la "valeur de MAC".

9.1 Type AuthenticatedData

L'identifiant d'objet suivant identifie le type de contenu authenticated-data :

IDENTIFIANT D'OBJET id-ct-authData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 2 }

Le type de contenu authenticated-data devra avoir le type ASN.1 AuthenticatedData :

```
AuthenticatedData ::= SEQUENCE {
    version                 CMSVersion,
    originatorInfo [0]      IMPLICITE OriginatorInfo OPTIONNEL,
    recipientInfos          RecipientInfos,
    macAlgorithm            MessageAuthenticationCodeAlgorithm,
    digestAlgorithm [1]     DigestAlgorithmIdentifier OPTIONNEL,
    encapContentInfo        EncapsulatedContentInfo,
    authAttrs [2]           IMPLICITE AuthAttributes OPTIONNEL,
    mac                     MessageAuthenticationCode,
    unauthAttrs [3]         IMPLICITE UnauthAttributes OPTIONNEL }

AuthAttributes ::= REGLER TAILLE (1..MAX) DE Attribute

UnauthAttributes ::= REGLER TAILLE (1..MAX) DE Attribute

MessageAuthenticationCode ::= CHAINE D'OCTETS
```

Les champs du type AuthenticatedData ont les significations suivantes :

version est le numéro de version de la syntaxe. La version DOIT être allouée comme suit :

```
SI (originatorInfo est présent) ET
    ((des certificats avec un type de other sont présents) OU
    (des crl avec un type de other sont présents))
ALORS version est 3
AUTREMENT
    SI ((originatorInfo est présent) ET
        (un certificat d'attribut de version 2 est présent))
    ALORS version est 1
    AUTREMENT version est 0
```

originatorInfo donne facultativement des informations sur l'origine. Il n'est présent qui si c'est exigé par l'algorithme de gestion de clé. Il PEUT contenir des certificats, des certificats d'attribut, et des CRL, comme défini au paragraphe 6.1.

recipientInfos est une collection d'informations par receveur, comme défini au paragraphe 6.1. Il DOIT y avoir au moins un élément dans la collection.

macAlgorithm est un identifiant d'algorithme de code d'authentification de message (MAC). Il identifie l'algorithme de MAC, ainsi que tous paramètres associés, utilisé par l'origine. Le placement du champ macAlgorithm facilite le traitement en un seul passage par le receveur.

digestAlgorithm identifie l'algorithme de résumé de message, et tous paramètres associés, utilisé pour calculer un résumé de message sur le contenu encapsulé si des attributs authentifiés sont présents. Le processus de résumé de message est décrit au paragraphe 9.2. Le placement du champ digestAlgorithm facilite le traitement en un seul passage par le receveur. Si le champ digestAlgorithm est présent, le champ authAttrs DOIT aussi être présent.

encapContentInfo est le contenu qui est authentifié, comme défini au paragraphe 5.2.

authAttrs est une collection d'attributs authentifiés. La structure authAttrs est facultative, mais elle DOIT être présente si le type de contenu de la valeur EncapsulatedContentInfo qui va être authentifiée n'est pas id-data. Si le champ authAttrs est présent, le champ digestAlgorithm DOIT alors être aussi présent. La structure AuthAttributes DOIT être codée en DER, même si le reste de la structure est codé en BER. Des types d'attribut utiles sont définis à la Section 11. Si le champ authAttrs est présent, il DOIT contenir, au minimum, les deux attributs suivants :

Un attribut type de contenu ayant comme valeur le type de contenu de la valeur EncapsulatedContentInfo en cours d'authentification. Le paragraphe 11.1 définit l'attribut type de contenu.

Un attribut résumé de message, ayant comme valeur le résumé de message du contenu. Le paragraphe 11.2 définit l'attribut résumé de message.

mac est le code d'authentification du message.

unauthAttrs est une collection d'attributs qui ne sont pas authentifiés. Le champ est facultatif. À ce jour, aucun attributs n'été défini pour être utilisé comme attribut non authentifié, mais d'autres types d'attribut utiles sont définis à la Section 11.

9.2 Génération du MAC

Le processus de calcul du MAC calcule un code d'authentification de message (MAC) sur le contenu à authentifier ou sur un résumé de message du contenu à authentifier avec les attributs authentifiés de l'origine.

Si le champ authAttrs est absent, l'entrée du processus de calcul du MAC est la valeur de la CHAINE D'OCTETS encapContentInfo eContent. Seuls les octets comprenant la valeur de la CHAINE D'OCTETS eContent sont entrés dans l'algorithme de MAC ; les octets de l'étiquette et de longueur sont omis. Cela présente l'avantage que la longueur du contenu à authentifier n'a pas besoin d'être connue à l'avance du processus de génération du MAC.

Si le champ authAttrs est présent, l'attribut type de contenu (comme décrit au paragraphe 11.1) et l'attribut résumé de message (comme décrit au paragraphe 11.2) DOIVENT être inclus, et l'entrée du processus de calcul de MAC est le codage en DER de authAttrs. Un codage distinct du champ authAttrs est effectué pour le calcul du résumé de message. L'étiquette IMPLICITE [2] n'est pas utilisée dans le champ authAttrs pour le codage en DER, et on utilise plutôt une étiquette EXPLICITE ENSEMBLE DE. C'est-à-dire que le codage en DER de l'étiquette ENSEMBLE DE est à inclure dans le calcul du résumé de message avec les octets de longueur et de contenu de la valeur de authAttrs plutôt que l'étiquette IMPLICITE [2].

Le processus de calcul de résumé de message calcule un résumé de message sur le contenu à authentifier. L'entrée initiale au processus de calcul de résumé de message est la "valeur" du contenu encapsulé à authentifier. Précisément, l'entrée est la CHAINE D'OCTETS encapContentInfo eContent à laquelle le processus d'authentification est appliqué. Seuls les octets comprenant la valeur de la CHAINE D'OCTETS encapContentInfo eContent sont entrés dans l'algorithme de résumé de message, et non l'étiquette ou les octets de longueur. Cela présente l'avantage que la longueur du contenu à authentifier n'a pas besoin d'être connue à l'avance. Bien que les octets de longueur et de l'étiquette CHAINE D'OCTETS encapContentInfo eContent ne soient pas inclus dans le calcul du résumé de message, ils sont quand même protégés par d'autres moyens. Les octets de longueur sont protégés par la nature de l'algorithme de résumé de message car il est impossible par le calcul de trouver deux contenus distincts de n'importe quelle longueur qui aient le même résumé de message.

L'entrée du processus de calcul du MAC comporte les données d'entrée du MAC, définies ci-dessus, et une clé d'authentification portée dans une structure recipientInfo. Les détails du calcul du MAC dépendent de l'algorithme de MAC employé (par exemple, un code d'authentification de message haché (HMAC, *Hashed Message Authentication Code*)). L'identifiant d'objet, avec tous ses paramètres, qui spécifie l'algorithme de MAC employé par l'origine est porté dans le champ macAlgorithm. La valeur de MAC générée par l'origine est codée comme une CHAINE D'OCTETS et portée dans le champ mac.

9.3 Vérification de MAC

L'entrée du processus de vérification du MAC comporte les données d'entrée (déterminées par la présence ou l'absence du champ authAttrs, comme défini en 9.2) et la clé d'authentification convoyée dans recipientInfo. Les détails du processus de vérification du MAC dépendent de l'algorithme de MAC employé.

Le receveur NE DOIT PAS s'appuyer sur des valeurs de MAC ou des valeurs de résumé de message calculées par l'origine. Le contenu est authentifié comme décrit au paragraphe 9.2. Si l'origine inclut des attributs authentifiés, le contenu de authAttrs est authentifié comme décrit au paragraphe 9.2. Pour que l'authentification réussisse, la valeur du MAC calculée par le receveur DOIT être la même que celle du champ mac. De même, pour que l'authentification réussisse lorsque le champ authAttrs est présent, la valeur du contenu du résumé de message calculée par le receveur DOIT être la même que la valeur du résumé de message incluse dans l'attribut message-digest authAttrs.

Si AuthenticatedData inclut un authAttrs, la valeur de l'attribut type de contenu DOIT correspondre à la valeur de AuthenticatedData encapContentInfo eContentType.

10. Types utiles

La présente section est divisée en deux parties. La première définit l'identifiant d'algorithmes, et la seconde définit les autres types utiles.

10.1 Types d'identifiant d'algorithme

Tous les identifiants d'algorithme ont le même type: AlgorithmIdentifier. La définition de AlgorithmIdentifier est tirée de la Recommandation UIT-T X.509 [X.509-88].

Il y a de nombreuses solutions de remplacement pour chaque type d'algorithme.

10.1.1 DigestAlgorithmIdentifier

Le type DigestAlgorithmIdentifier identifie un algorithme de résumé de message. Parmi les exemples on citera SHA-1, MD2, et MD5. Un algorithme de résumé de message transpose une chaîne d'octets (le contenu) en une autre chaîne d'octets (le résumé de message).

DigestAlgorithmIdentifier ::= AlgorithmIdentifier

10.1.2 SignatureAlgorithmIdentifier

Le type SignatureAlgorithmIdentifier identifie un algorithme de signature, et il peut aussi identifier un algorithme de résumé de message. Parmi les exemples on citera RSA, DSA, DSA avec SHA-1, ECDSA, et ECDSA avec SHA-256. Un algorithme de signature prend en charge les opérations de génération et vérification de signature. L'opération de génération de signature utilise le résumé de message et la clé privée du signataire pour générer une valeur de signature. L'opération de vérification de signature utilise le résumé de message et la clé publique du signataire pour déterminer si une valeur de signature est ou non valide. Le contexte détermine l'opération à effectuer.

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

10.1.3 KeyEncryptionAlgorithmIdentifier

Le type KeyEncryptionAlgorithmIdentifier identifie un algorithme de chiffrement de clé utilisé pour chiffrer une clé de chiffrement de contenu. L'opération de chiffrement transpose une chaîne d'octets (la clé) en une autre chaîne d'octets (la clé chiffrée) sous le contrôle d'une clé de chiffrement de clé. L'opération de déchiffrement est l'inverse de l'opération de chiffrement. Le contexte détermine quelle opération effectuer.

Les détails du chiffrement et du déchiffrement dépendent de l'algorithme de gestion de clé utilisé. Le transport de clé, l'accord de clé, les clés de chiffrement de clé symétriques à distribution préalable, et les clés de chiffrement de clé symétriques déduites de mots de passe sont pris en charge.

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

10.1.4 ContentEncryptionAlgorithmIdentifier

Le type ContentEncryptionAlgorithmIdentifier identifie un algorithme de chiffrement de contenu. On citera en exemple Triple-DES et RC2. Un algorithme de chiffrement de contenu prend en charge les opérations de chiffrement et de déchiffrement. L'opération de chiffrement transpose une chaîne d'octets (le texte en clair) en une autre chaîne d'octets (le texte chiffré) sous le contrôle d'une clé de chiffrement de contenu. L'opération de déchiffrement est l'inverse de l'opération de chiffrement. Le contexte détermine quelle opération est effectuée.

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

10.1.5 MessageAuthenticationCodeAlgorithm

Le type MessageAuthenticationCodeAlgorithm identifie un algorithme de code d'authentification de message (MAC). Des exemples en sont DES-MAC et HMAC-SHA-1. Un algorithme de MAC prend en charge les opérations de génération et vérification. Les opérations de génération et de vérification de MAC utilisent la même clé symétrique. Le contexte détermine quelle opération effectuer.

MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier

10.1.6 KeyDerivationAlgorithmIdentifier

Le type KeyDerivationAlgorithmIdentifier est spécifié dans la [RFC3211]. La définition de KeyDerivationAlgorithmIdentifier est répétée ici dans un souci de complétude.

Les algorithmes de déduction de clé convertissent un mot de passe ou valeur de secret partagé en une clé de chiffrement de clé.

```
KeyDerivationAlgorithmIdentifier ::= AlgorithmIdentifier
```

10.2 Autres types utiles

Cette section définit les types qui sont utilisés dans d'autres endroits dans ce document. Les types ne sont pas énumérés dans un ordre particulier.

10.2.1 RevocationInfoChoices

Le type RevocationInfoChoices donne un ensemble de solutions de remplacement pour des informations d'état de révocation. Il est prévu que l'ensemble contienne des informations suffisantes pour déterminer si les certificats et certificats d'attribut auxquels l'ensemble est associé sont révoqués. Cependant, il PEUT y avoir plus d'informations d'état de révocation que nécessaire ou il PEUT y avoir moins d'informations d'état de révocation que nécessaire. Les listes de révocation de certificat (CRL, *Certificate Revocation List*) de la Recommandation UIT-T X.509 [X.509-97] sont les principales sources d'informations d'état de révocation, mais tout autre format d'informations de révocation peut être accepté. La solution de remplacement OtherRevocationInfoFormat est fournie pour la prise en charge de tous les autres formats d'information de révocation sans autre modification de la CMS. Par exemple, les réponses du protocole d'état de certificat en ligne (OCSP, *Online Certificate Status Protocol*) [RFC2560] peuvent être prises en charge en utilisant OtherRevocationInfoFormat.

La liste des certificats peut contenir une CRL, une liste de révocation d'autorité (ARL, *Authority Revocation List*), une CRL par différence, ou une liste de révocation de certificat d'attribut. Toutes ces listes partagent une syntaxe commune.

Le type CertificateList donne une liste de révocation de certificat (CRL). Les CRL sont spécifiées dans la Recommandation UIT-T X.509 [X.509-97], et elles ont des profils d'utilisation dans l'Internet décrits par la [RFC5280].

La définition de CertificateList est tirée de X.509.

```
RevocationInfoChoices ::= ENSEMBLE DE RevocationInfoChoice

RevocationInfoChoice ::= CHOIX {
    crl                         CertificateList,
    other [1]                   IMPLICIT OtherRevocationInfoFormat }

OtherRevocationInfoFormat ::= SEQUENCE {
    otherRevInfoFormat          IDENTIFIANT D'OBJET,
    otherRevInfo                TOUT DEFINI PAR otherRevInfoFormat }
```

10.2.2 CertificateChoices

Le type CertificateChoices donne un certificat PKCS n° 6 étendu [PKCS#6], un certificat X.509, un certificat d'attribut X.509 version 1 (ACv1) [X.509-97], un certificat d'attribut X.509 version 2 (ACv2) [X.509-00], ou tout autre format de certificat. Le certificat PKCS n° 6 étendu est obsolète. Le certificat PKCS n° 6 est inclus pour la rétro compatibilité, et les certificats PKCS n° 6 NE DEVRAIENT PAS être utilisés. Le certificat ACv1 est aussi obsolète. ACv1 est inclus pour la rétro compatibilité, et ACv1 NE DEVRAIT PAS être utilisé. Le profil Internet des certificats X.509 est spécifié dans "Profil de certificat d'infrastructure de clé publique X.509 et de liste de révocation de certificat (CRL) pour l'Internet" [RFC5280]. Le profil Internet de ACv2 est spécifié dans "Profil de certificat d'attribut Internet pour l'autorisation" [RFC3281]. La solution de remplacement OtherCertificateFormat est fournie pour prendre en charge tout autre format de certificat sans autre modification à la CMS.

La définition de Certificate est tirés de X.509.

Les définitions de AttributeCertificate sont tirées de X.509-1997 et X.509-2000. La définition tirée de X.509-1997 est allouée à AttributeCertificateV1 (voir au paragraphe 12.2) et la définition de X.509-2000 est allouée à AttributeCertificateV2.

```
CertificateChoices ::= CHOIX {
    certificate                 Certificate,
    extendedCertificate [0] I           MPLICITE ExtendedCertificate,       -- Obsolète
    v1AttrCert [1]                      IMPLICIT AttributeCertificateV1,    -- Obsolète
    v2AttrCert [2]              IMPLICIT AttributeCertificateV2,
    other [3]                   IMPLICIT OtherCertificateFormat }

OtherCertificateFormat ::= SEQUENCE {
    otherCertFormat             IDENTIFIANT D'OBJET,
    otherCert                   TOUT DEFINI PAR otherCertFormat }
```

10.2.3 CertificateSet

Le type CertificateSet donne un ensemble de certificats. Il est prévu que l'ensemble soit suffisant pour contenir les chemins de certification à partir d'une "racine" reconnue ou d'une "autorité de certification de niveau supérieur" vers tous les certificats d'envoyeur à qui l'ensemble est associé. Cependant, il peut y avoir plus de certificats que nécessaire, ou il PEUT y en avoir moins que nécessaire.

La signification précise de "chemin de certification" sort du domaine d'application du présent document. Cependant, la [RFC5280] donne une définition pour les certificats X.509. Certaines applications peuvent imposer des limites supérieures à la longueur d'un chemin de certification ; d'autres peuvent mettre en application une certaine relation entre les sujets et les producteurs de certificats au sein d'un chemin de certification.

```
CertificateSet ::= ENSEMBLE DE CertificateChoices
```

10.2.4 IssuerAndSerialNumber

Le type IssuerAndSerialNumber identifie un certificat, et par là, une entité et une clé publique, par le nom distinctif du producteur du certificat et un numéro de série du certificat spécifique du producteur.

La définition de Name est tirée de la Recommandation UIT-T X.501 [X.501-88], et la définition de CertificateSerialNumber est tirée de la Recommandation UIT-T X.509 [X.509-97].

```
IssuerAndSerialNumber ::= SEQUENCE { Nom du producteur, serialNumber CertificateSerialNumber }

CertificateSerialNumber ::= ENTIER
```

10.2.5 CMSVersion

Le type CMSVersion donne un numéro de version de la syntaxe, pour la compatibilité avec de futures révisions de la présente spécification.

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

10.2.6 UserKeyingMaterial

Le type UserKeyingMaterial donne une syntaxe pour le matériel de clé d'utilisateur (UKM, *user keying material*). Certains algorithmes d'accord de clé exigent que les UKM assurent qu'une clé différente sera générée à chaque fois que les deux mêmes parties génèrent une paire de clés. L'envoyeur fournit un UKM à utiliser avec un algorithme d'accord de clé spécifique.

```
UserKeyingMaterial ::= CHAINE D'OCTETS
```

10.2.7 OtherKeyAttribute

Le type OtherKeyAttribute donne une syntaxe pour l'inclusion d'autres attributs de clé qui permettent au receveur de choisir la clé utilisée par l'envoyeur. L'objet Identifiant d'attribut doit être enregistré avec la syntaxe de l'attribut lui-même. L'utilisation de cette structure devrait être évitée car elle peut empêcher l'interopérabilité.

```
OtherKeyAttribute ::= SEQUENCE {
    keyAttrId                   IDENTIFIANT D'OBJET,
    keyAttr                     TOUT DEFINI PAR keyAttrId OPTIONNEL }
```

11. Attributs utiles

La présente section définit des attributs qui peuvent être utilisés avec des données signées, des données enveloppées, des données chiffrées, ou des données authentifiées. La syntaxe d'attribut est compatible avec celle de [X.501-88] et de la [RFC5280]. Certains des attributs définis dans la présente section ont été à l'origine définis dans [PKCS#9] ; d'autres ont été à l'origine définis dans une version précédente [RFC2630] de la présente spécification. Les attributs ne sont pas données dans un ordre particulier.

Des attributs sont définis dans de nombreux endroits, en particulier dans la spécification de message S/MIME version 3.1 de la [RFC3851] et les services de sécurité améliorée S/MIME [RFC2634], qui incluent aussi des recommandations sur le placement de ces attributs.

11.1 Type de contenu

Le type d'attribut Type de contenu spécifie le type de contenu des ContentInfo au sein de données signées ou de données authentifiées. Le type d'attribut type de contenu DOIT être présent chaque fois que des attributs signés sont présents dans des données signées ou que des attributs authentifiés sont présents dans des données authentifiées. La valeur de l'attribut type de contenu DOIT correspondre à la valeur de encapContentInfo eContentType dans les données signées ou authentifiées.

L'attribut type de contenu DOIT être un attribut signé ou un attribut authentifié ; il NE DOIT PAS être un attribut non signé, un attribut non authentifié, ou un attribut non protégé.

L'identifiant d'objet suivant identifie l'attribut type de contenu :

IDENTIFIANT D'OBJET id-contentType ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }

Les valeurs de l'attribut Content-type ont le type ASN.1 ContentType :

ContentType ::= IDENTIFIANT D'OBJET

Bien que la syntaxe soit définie comme un ENSEMBLE DE AttributeValue, un attribut type de contenu DOIT avoir une seule valeur d'attribut ; zéro ou plusieurs instances de AttributeValue ne sont pas permises.

Les syntaxes de SignedAttributes et AuthAttributes sont chacune définies comme un ENSEMBLE DE Attributes. Le SignedAttributes dans un signerInfo NE DOIT PAS inclure plusieurs instances de l'attribut type de contenu. De même, le AuthAttributes dans un AuthenticatedData NE DOIT PAS inclure plusieurs instances de l'attribut type de contenu.

11.2 Résumé de message

L'attribut message-digest spécifie le résumé de message de la CHAINE D'OCTETS encapContentInfo eContent en cours de signature dans les données signées (voir au paragraphe 5.4) ou d'authentification dans les données authentifiées (voir au paragraphe 9.2). Pour les données signées, le résumé de message est calculé en utilisant l'algorithme de résumé de message de l'envoyeur. Pour les données authentifiées, le résumé de message est calculé en utilisant l'algorithme de résumé de message de l'origine.

Au sein des données signées, le type d'attribut résumé de message signé DOIT être présent lorsque il y a un ou des attributs signés présents. Au sein de données authentifiées, le type authentifié d'attribut résumé de message DOIT être présent lorsque sont présents un ou des attributs authentifiés.

L'attribut message-digest DOIT être un attribut signé ou un attribut authentifié ; il NE DOIT PAS être un attribut non signé, un attribut non authentifié, ou un attribut non protégé.

L'identifiant d'objet suivant identifie l'attribut message-digest :

IDENTIFIANT D'OBJET id-messageDigest ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }

La valeur de l'attribut message-digest a le type ASN.1 MessageDigest :

MessageDigest ::= CHAINE D'OCTETS

Un attribut message-digest DOIT avoir une seule valeur d'attribut, même si la syntaxe est définie comme un ENSEMBLE DE AttributeValue. Il NE DOIT PAS y avoir zéro ou plusieurs instances de AttributeValue présentes.

La syntaxe de SignedAttributes et la syntaxe de AuthAttributes sont chacune définies comme un ENSEMBLE DE Attributes. Le SignedAttributes dans une signerInfo DOIT inclure seulement une instance de l'attribut message-digest. De même, le AuthAttributes dans un AuthenticatedData DOIT inclure seulement une instance de l'attribut message-digest.

11.3 Heure de signature

Le type d'attribut signing-time spécifie l'heure à laquelle le signataire a effectué (délibérément) le processus de signature. Le type d'attribut signing-time est destiné à être utilisé dans des données signées.

L'attribut signing-time DOIT être un attribut signé ou un attribut authentifié ; il NE DOIT PAS être un attribut non signé, un attribut non authentifié, ou un attribut non protégé.

L'identifiant d'objet suivant identifie l'attribut signing-time :

IDENTIFIANT D'OBJET id-signingTime ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }

Les valeurs d'attribut Signing-time ont un type ASN.1 SigningTime :

```
SigningTime ::= Time

Time ::= CHOIX {
    utcTime             UTCTime,
    generalizedTime     GeneralizedTime }
```

Note : La définition de Time correspond à celle spécifiée dans la version 1997 de X.509 [X.509-97].

Les dates entre le 1e janvier 1950 et le 31 décembre 2049 (inclus) DOIVENT être codées en UTCTime. Toutes les dates avec des valeurs d'année avant 1950 ou après 2049 DOIVENT être codées en GeneralizedTime.

Les valeurs de UTCTime DOIVENT être exprimées en temps universel coordonné (anciennement appelé Heure moyenne de Greenwich (GMT) et heure Zoulou) et DOIVENT inclure les secondes (c'est-à-dire que les heures sont AAMMJJHHMMSSZ) même lorsque le nombre de secondes est zéro. Minuit DOIT être représenté par "AAMMJJ000000Z". L'information sur le siècle est implicite ; le siècle DOIT être déterminé comme suit :

Lorsque AA est supérieur ou égal à 50, l'année DOIT être interprétée comme 19AA ; et

Lorsque AA est inférieur à 50, l'année DOIT être interprétée comme 20AA.

Les valeurs de GeneralizedTime DOIVENT être exprimées en temps universel coordonné et DOIVENT inclure les secondes (c'est-à-dire que les heures sont AAAAMMJJHHMMSSZ) même lorsque le nombre de secondes est zéro. Les valeurs de GeneralizedTime NE DOIVENT PAS inclure de fraction de seconde.

Un attribut signing-time DOIT avoir une seule valeur d'attribut, même si la syntaxe est définie comme un ENSEMBLE DE AttributeValue. Il NE DOIT PAS y avoir zéro ou plusieurs instances de AttributeValue présentes.

La syntaxe de SignedAttributes et la syntaxe de AuthAttributes sont chacune définies comme un ENSEMBLE DE Attributes. Le SignedAttributes dans un signerInfo NE DOIT PAS inclure plusieurs instances de l'attribut signing-time. De même, le AuthAttributes dans un AuthenticatedData NE DOIT PAS inclure plusieurs instances de l'attribut signing-time.

Aucune exigence n'est imposée sur l'exactitude de l'heure de signature, et l'acceptation d'une heure de signature prétendue est à la discrétion du receveur. On s'attend cependant à ce que certains signataires, comme les serveurs d'horodatage, soient implicitement de confiance.

11.4 Contreseing

Le type d'attribut contreseing spécifie une ou plusieurs signatures sur les octets de contenu de la CHAINE D'OCTETS de signature dans une valeur SignerInfo des données signées. C'est-à-dire que le résumé de message est calculé sur les octets qui comportent la valeur de la CHAINE D'OCTETS, et ni l'étiquette ni les octets de longueur ne sont inclus. Donc, le type d'attribut contreseing contresigne (signe à la suite) une autre signature.

L'attribut contreseing DOIT être un attribut non signé ; il NE DOIT PAS être un attribut signé, un attribut authentifié, un attribut non authentifié, ou un attribut non protégé.

L'identifiant d'objet suivant identifie l'attribut contreseing :

IDENTIFIANT D'OBJET id-countersignature ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 6 }

Les valeurs de l'attribut Countersignature ont le type ASN.1 Countersignature :

Countersignature ::= SignerInfo

Les valeurs de contreseing ont la même signification que les valeurs de SignerInfo pour les signatures ordinaires, sauf que :

1. Le champ signedAttributes NE DOIT PAS contenir un attribut type de contenu ; il n'y a pas de type de contenu pour les contreseings.

2. Le champ signedAttributes DOIT contenir un attribut message-digest si il contient d'autres attributs.

3. L'entrée au processus de résumé de message sont les octets de contenu du codage en DER du champ signatureValue de la valeur SignerInfo à laquelle l'attribut est associé.

Un attribut contreseing peut avoir plusieurs valeurs d'attribut. La syntaxe est définie comme un ENSEMBLE DE AttributeValue, et il DOIT y avoir une ou plusieurs instances de AttributeValue présentes.

La syntaxe de UnsignedAttributes est définie comme un ENSEMBLE DE Attributes. Le UnsignedAttributes dans un signerInfo peut inclure plusieurs instances de l'attribut contreseing.

Un contreseing, comme il a le type SignerInfo, peut lui-même contenir un attribut contreseing. Donc, il est possible de construire une série d'une longueur arbitraire de contreseings.

12. Modules ASN.1

La section 12.1 contient le module ASN.1 pour la CMS, et la section 12.2 contient le module ASN.1 pour le certificat d'attribut version 1.

12.1 Module ASN.1 de CMS

```
CryptographicMessageSyntax2004
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) }

ÉTIQUETTES IMPLICITES DE DÉFINITIONS ::=
DÉBUT

-- EXPORTE Tout
-- Les types et valeurs définies dans ce module sont exportées pour utilisation
-- dans les autres modules ASN.1. D'autres applications peuvent les utiliser pour leur propres besoins.

IMPORTE

-- Imports de la [RFC5280], Appendice A.1
        AlgorithmIdentifier, Certificate, CertificateList,
        CertificateSerialNumber, Name
          DE PKIX1Explicit88
              { iso(1) identified-organization(3) dod(6)
                internet(1) security(5) mechanisms(5) pkix(7)
```

```
                mod(0) pkix1-explicit(18) }

-- Imports de la [RFC3281], Appendice B
        AttributeCertificate
          DE PKIXAttributeCertificate
              { iso(1) identified-organization(3) dod(6)
                internet(1) security(5) mechanisms(5) pkix(7)
                mod(0) attribute-cert(12) }

-- Imports de l'Appendice B du présent document
        AttributeCertificateV1
          DE AttributeCertificateVersion1
              { iso(1) member-body(2) us(840) rsadsi(113549)
                pkcs(1) pkcs-9(9) smime(16) modules(0)
                v1AttrCert(15) } ;

-- Syntaxe de message cryptographique

ContentInfo ::= SEQUENCE {
    contentType         ContentType,
    content [0]         EXPLICITE TOUT DEFINI PAR contentType }

ContentType ::= IDENTIFIANT D'OBJET

SignedData ::= SEQUENCE {
    version             CMSVersion,
    digestAlgorithms    DigestAlgorithmIdentifiers,
    encapContentInfo    EncapsulatedContentInfo,
    certificates [0]    IMPLICITE CertificateSet OPTIONNEL,
    crls [1]            IMPLICITE RevocationInfoChoices OPTIONNEL,
    signerInfos         SignerInfos }

DigestAlgorithmIdentifiers ::= ENSEMBLE DE DigestAlgorithmIdentifier

SignerInfos ::= ENSEMBLE DE SignerInfo
EncapsulatedContentInfo ::= SEQUENCE {
    eContentType        ContentType,
    eContent [0]        EXPLICITE CHAINE D'OCTETS OPTIONNEL }

SignerInfo ::= SEQUENCE {
    version             CMSVersion,
    sid                 SignerIdentifier,
    digestAlgorithm     DigestAlgorithmIdentifier,
    signedAttrs [0]     MPLICIT SignedAttributes OPTIONNEL,
    signatureAlgorithm  SignatureAlgorithmIdentifier,
    signature           SignatureValue,
    unsignedAttrs [1]   IMPLICIT UnsignedAttributes OPTIONNEL }

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

SignedAttributes ::= REGLE TAILLE (1..MAX) DE Attribute

UnsignedAttributes ::= REGLE TAILLE (1..MAX) DE Attribute

Attribute ::= SEQUENCE {
    attrType            IDENTIFIANT D'OBJET,
    attrValues          ENSEMBLE DE AttributeValue }

AttributeValue ::= TOUT

SignatureValue ::= CHAINE D'OCTETS
```

```
EnvelopedData ::= SEQUENCE {
    version                 CMSVersion,
    originatorInfo [0]      IMPLICIT OriginatorInfo OPTIONNEL,    recipientInfos RecipientInfos,
    encryptedContentInfo    EncryptedContentInfo,
    unprotectedAttrs [1]    IMPLICITE UnprotectedAttributes OPTIONNEL }

OriginatorInfo ::= SEQUENCE {
    certs [0]       IMPLICITE CertificateSet OPTIONNEL,
    crls [1]        IMPLICITE RevocationInfoChoices OPTIONNEL }

RecipientInfos ::= REGLE TAILLE (1..MAX) DE RecipientInfo

EncryptedContentInfo ::= SEQUENCE {
    contentType                 ContentType,
    contentEncryptionAlgorithm  ContentEncryptionAlgorithmIdentifier,
    encryptedContent [0]        IMPLICITE EncryptedContent OPTIONNEL }

EncryptedContent ::= CHAINE D'OCTETS

UnprotectedAttributes ::= REGLE TAILLE (1..MAX) DE Attribute

RecipientInfo ::= CHOIX {
    ktri            KeyTransRecipientInfo,
    kari [1]        KeyAgreeRecipientInfo,
    kekri [2]       KEKRecipientInfo,
    pwri [3]        PasswordRecipientInfo,
    ori [4]         OtherRecipientInfo }

EncryptedKey ::= CHAINE D'OCTETS

KeyTransRecipientInfo ::= SEQUENCE {
    version                     CMSVersion,                 -- toujours réglé à 0 ou 2
    rid                         RecipientIdentifier,
    keyEncryptionAlgorithm      KeyEncryptionAlgorithmIdentifier,
    encryptedKey                EncryptedKey }

RecipientIdentifier ::= CHOIX {
    issuerAndSerialNumber       IssuerAndSerialNumber,
    subjectKeyIdentifier [0]    SubjectKeyIdentifier }

KeyAgreeRecipientInfo ::= SEQUENCE {
    version                     CMSVersion,                 -- toujours réglé à 3
    originator [0]              EXPLICITE OriginatorIdentifierOrKey,
    ukm [1]                     EXPLICITE UserKeyingMaterial OPTIONNEL,
    keyEncryptionAlgorithm      KeyEncryptionAlgorithmIdentifier,
    recipientEncryptedKeys      RecipientEncryptedKeys }

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

OriginatorPublicKey ::= SEQUENCE {
    algorithm       AlgorithmIdentifier,
    publicKey       CHAINE BINAIRE }

RecipientEncryptedKeys ::= SEQUENCE DE RecipientEncryptedKey

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

```
KeyAgreeRecipientIdentifier ::= CHOIX {
    issuerAndSerialNumber   IssuerAndSerialNumber,
    rKeyId [0] IMPLICIT     RecipientKeyIdentifier }

RecipientKeyIdentifier ::= SEQUENCE {
    subjectKeyIdentifier    SubjectKeyIdentifier,
    date GeneralizedTime    OPTIONNEL,
    other OtherKeyAttribute OPTIONNEL }

SubjectKeyIdentifier ::= CHAINE D'OCTETS

KEKRecipientInfo ::= SEQUENCE {
    version                 CMSVersion,                 -- toujours réglé à 4
    kekid                   KEKIdentifier,
    keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
    encryptedKey            EncryptedKey }

KEKIdentifier ::= SEQUENCE {
    keyIdentifier           CHAINE D'OCTETS,
    date GeneralizedTime    OPTIONNEL,
    other OtherKeyAttribute OPTIONNEL }

PasswordRecipientInfo ::= SEQUENCE {
    version                 CMSVersion,                 -- toujours réglé à 0
    keyDerivationAlgorithm [0]KeyDerivationAlgorithmIdentifier OPTIONNEL,
    keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
    encryptedKey            EncryptedKey }

OtherRecipientInfo ::= SEQUENCE {
    oriType         IDENTIFIANT D'OBJET,
    oriValue        ANY DEFINED BY oriType }

DigestedData ::= SEQUENCE {
    version             CMSVersion,
    digestAlgorithm     DigestAlgorithmIdentifier,
    encapContentInfo    EncapsulatedContentInfo,
    digest              Digest }

Digest ::= CHAINE D'OCTETS

EncryptedData ::= SEQUENCE {
    version             CMSVersion,
    encryptedContentInfo    EncryptedContentInfo,
    unprotectedAttrs [1]    IMPLICIT UnprotectedAttributes OPTIONNEL }

AuthenticatedData ::= SEQUENCE {
    version             CMSVersion,
    originatorInfo [0]  IMPLICIT OriginatorInfo OPTIONNEL,
    recipientInfos      RecipientInfos,
    macAlgorithm        MessageAuthenticationCodeAlgorithm,
    digestAlgorithm [1] DigestAlgorithmIdentifier OPTIONNEL,
    encapContentInfo    EncapsulatedContentInfo,
    authAttrs [2]       IMPLICIT AuthAttributes OPTIONNEL,
    mac                 MessageAuthenticationCode,
    unauthAttrs [3]     IMPLICIT UnauthAttributes OPTIONNEL }

AuthAttributes ::= REGLE TAILLE (1..MAX) DE Attribute

UnauthAttributes ::= REGLE TAILLE (1..MAX) DE Attribute

MessageAuthenticationCode ::= CHAINE D'OCTETS

DigestAlgorithmIdentifier ::= AlgorithmIdentifier
```

```
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier

KeyDerivationAlgorithmIdentifier ::= AlgorithmIdentifier

RevocationInfoChoices ::= ENSEMBLE DE RevocationInfoChoice

RevocationInfoChoice ::= CHOIX {
   crl           CertificateList,
   other [1]     IMPLICIT OtherRevocationInfoFormat }

OtherRevocationInfoFormat ::= SEQUENCE {
   otherRevInfoFormat    IDENTIFIANT D'OBJET,
   otherRevInfo          TOUT DEFINI PAR otherRevInfoFormat }

CertificateChoices ::= CHOIX {
   certificate           Certificate,
   extendedCertificate [0] IMPLICIT ExtendedCertificate,          -- Obsolète
   v1AttrCert [1]        IMPLICIT AttributeCertificateV1,         -- Obsolète
   v2AttrCert [2]        IMPLICIT AttributeCertificateV2,
   other [3]             IMPLICIT OtherCertificateFormat }

AttributeCertificateV2 ::= AttributeCertificate

OtherCertificateFormat ::= SEQUENCE {
   otherCertFormat       IDENTIFIANT D'OBJET,
   otherCert             TOUT DEFINI PAR otherCertFormat }

CertificateSet ::= ENSEMBLE DE CertificateChoices

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

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

UserKeyingMaterial ::= CHAINE D'OCTETS

OtherKeyAttribute ::= SEQUENCE {
   IDENTIFIANT D'OBJET   keyAttrId ,
   keyAttr               TOUT DEFINI PAR keyAttrId OPTIONNEL }

-- Identifiants d'objets Type de contenu

IDENTIFIANT D'OBJET id-ct-contentInfo ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
                                            smime(16) ct(1) 6 }

IDENTIFIANT D'OBJET id-data ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 1 }

IDENTIFIANT D'OBJET id-signedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }

IDENTIFIANT D'OBJET id-envelopedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3 }

IDENTIFIANT D'OBJET id-digestedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 5 }

IDENTIFIANT D'OBJET id-encryptedData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 6 }

IDENTIFIANT D'OBJET id-ct-authData ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
                                         smime(16) ct(1) 2 }
```

```
-- Attributs de la CMS

MessageDigest ::= CHAINE D'OCTETS

SigningTime  ::= Time

Time ::= CHOIX { utcTime UTCTime, generalTime GeneralizedTime }

Countersignature ::= SignerInfo

-- Identifiants d'objet Attribute

id-contentType IDENTIFIANT D'OBJET ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }

id-messageDigest IDENTIFIANT D'OBJET ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }

id-signingTime IDENTIFIANT D'OBJET ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }

id-contreseing IDENTIFIANT D'OBJET ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 6 }

-- Syntaxe obsolète de certificat étendu de PKCS n° 6

ExtendedCertificateOrCertificate ::= CHOIX {
    certificate                 Certificate,
    extendedCertificate [0]     IMPLICIT ExtendedCertificate }

ExtendedCertificate ::= SEQUENCE {
    extendedCertificateInfo     ExtendedCertificateInfo,
    signatureAlgorithm          SignatureAlgorithmIdentifier,
    signature                   Signature }

ExtendedCertificateInfo ::= SEQUENCE {
    version         CMSVersion,
    certificate     Certificate,
    attributes      UnauthAttributes }

Signature ::= CHAINE BINAIRE

FIN                                                    -- de CryptographicMessageSyntax2004
```

12.2 Module ASN.1 de certificat d'attribut version 1

```
AttributeCertificateVersion1
     { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) v1AttrCert(15) }

DEFINITIONS DES ÉTIQUETTES EXPLICITES ::=
DÉBUT

-- EXPORTE Tout

IMPORTE

 -- Imports de la [RFC5280], Appendice A.1
        AlgorithmIdentifier, Attribute, CertificateSerialNumber, Extensions, UniqueIdentifier
          DE PKIX1Explicit88
              { iso(1) identified-organization(3) dod(6)
                internet(1) security(5) mechanisms(5) pkix(7) mod(0) pkix1-explicit(18) }

-- Imports de la [RFC5280], Appendice A.2
        GeneralNames
```

```
        DE PKIX1Implicit88
            { iso(1) identified-organization(3) dod(6)
              internet(1) security(5) mechanisms(5) pkix(7) mod(0) pkix1-implicit(19) }

-- Imports de la [RFC3281], Appendice B
        AttCertValidityPeriod, IssuerSerial
          DE PKIXAttributeCertificate
            { iso(1) identified-organization(3) dod(6)
              internet(1) security(5) mechanisms(5) pkix(7) mod(0) attribute-cert(12) } ;

-- Définition extraite de [X.509-97], mais des noms de type différents sont utilisés pour éviter les collisions.

AttributeCertificateV1 ::= SEQUENCE {
    acInfo                  AttributeCertificateInfoV1,
    signatureAlgorithm      AlgorithmIdentifier,
    signature               CHAINE BINAIRE }

AttributeCertificateInfoV1 ::= SEQUENCE {
    version                   AttCertVersionV1 DEFAUT v1,
    subject                   CHOIX {
      baseCertificateID [0]   IssuerSerial,          -- associé à un certificat de clé publique
      subjectName [1]         GeneralNames },        -- associé à un nom
    issuer                    GeneralNames,
    signature                 AlgorithmIdentifier,
    serialNumber              CertificateSerialNumber,
    attCertValidityPeriod     AttCertValidityPeriod,
    attributes                SEQUENCE DE Attribute,
    issuerUniqueID            UniqueIdentifier OPTIONNEL,
    extensions                Extensions OPTIONNEL }

AttCertVersionV1 ::= ENTIER { v1(0) }

FIN                                                -- de AttributeCertificateVersion1
```

13. Références

13.1 Références normatives

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997.

[RFC3281] S. Farrell et R. Housley, "Profil de certificat d'attribut Internet pour l'autorisation", avril 2002. *(Remplacée par* RFC5755)

[RFC5280] D. Cooper et autres, "Profil de certificat d'infrastructure de clé publique X.509 et de liste de révocation de certificat (CRL) pour l'Internet", mai 2008. *(P.S.)*

[X.208-88] Recommandation CCITT X.208, "Spécification de la notation de syntaxe abstraite n° 1 (ASN.1)", 1988.

[X.209-88] Recommandation CCITT X.209, "Spécification des règles de codage de base pour la notation de syntaxe abstraite n° 1 (ASN.1)", 1988.

[X.501-88] Recommandation CCITT X.501, "L'annuaire – Modèles", 1988.

[X.509-88] Recommandation CCITT X.509, "L'annuaire – Cadre d'authentification", 1988.

[X.509-97] Recommandation UIT-T X.509, "L'annuaire – Cadre d'authentification", 1997.

[X.509-00] Recommandation UIT-T X.509, "L'annuaire – Cadre d'authentification", 2000.

13.2 Références pour information

[MSAC] Microsoft Development Network (MSDN) Library, "Authenticode", Livraison avril 2004.

[PKCS#6] RSA Laboratories. "PKCS #6: Extended-Certificate Syntax Standard, Version 1.5". novembre 1993.

[PKCS#9] RSA Laboratories. "PKCS #9: Selected Attribute Types, Version 1.1". novembre 1993.

[RFC2311] S. Dusse et autres, "Spécification de message S/MIME, version 2", mars 1998. *(Information)*

[RFC2313] B. Kaliski, "PKCS n° 1 : Chiffrement RSA version 1.5", mars 1998.

[RFC2315] B. Kaliski, "PKCS n° 7 : Syntaxe de message cryptographique, version 1.5", mars 1998. *(Information)*

[RFC2437] B. Kaliski et J. Staddon, "PKCS n° 1 : Spécifications de la cryptographie RSA version 2.0", octobre 1998. *(Obsolète, voir RFC 3447) (Information)*

[RFC2560] M. Myers, R. Ankney, A. Malpani, S. Galperin et C. Adams, "Protocole d'état de certificat en ligne d'infrastructure de clé publique X.509 pour l'Internet - OCSP", juin 1999. *(P.S.)*

[RFC2630] R. Housley, "Syntaxe de message cryptographique", juin 1999. *(Obsolète, voir 3369, 3370) (P.S.)*

[RFC2631] E. Rescorla, "Méthode d'accord de clé Diffie-Hellman", juin 1999. *(P.S.)*

[RFC2633] B. Rmasdell, "Spécification de message S/MIME version 3", juin 1999. (*Obsolète, voir* RFC3851) *(P.S.)*

[RFC2634] P. Hoffman, éd., "Services de sécurité améliorés pour S/MIME", juin 1999. (*MàJ par* RFC5035) *(P.S.)*

[RFC3211] P. Gutmann, "Chiffrement fondé sur le mot de passe pour CMS", décembre 2001. (*Obsolète, voir* RFC3370)

[RFC3369] R. Housley, "Syntaxe de message cryptographique (CMS)", août 2002. (*Obsolète, voir* RFC3852) *(P.S.)*

[RFC3370] R. Housley, "Algorithmes de syntaxe de message cryptographique (CMS)", août 2002. *(P.S.)*

[RFC3851] B. Ramsdell, "Spécification du message d'extensions de messagerie Internet multi-objets/sécurisé (S/MIME) version 3.1", juillet 2004. *(Remplacée par* RFC5751)

[RFC3852] R. Housley, "Syntaxe de message cryptographique (CMS)", juillet 2004. *(Remplacée par la présente RFC)*

[RFC4086] D. Eastlake 3rd, J. Schiller, S. Crocker, "Exigences d'aléa pour la sécurité", juin 2005. (BCP0106)

[RFC4853] R. Housley, "Syntaxe de message cryptographique (CMS) : précisions sur les signataires multiples", avril 2007. *(P.S.)*

14. Considérations pour la sécurité

La syntaxe de message cryptographique donne une méthode pour la signature numérique des données, le résumé des données, le chiffrement des données, et l'authentification des données.

Les mises en œuvre doivent protéger la clé privé du signataire. La compromission de la clé privée du signataire permet l'usurpation d'identité.

Les mises en œuvre doivent protéger la clé privée de gestion de clé, la clé de chiffrement de clé, et la clé ce chiffrement de contenu. La compromission d'une clé privée de gestion de clé ou d'une clé de chiffrement de clé peut déboucher sur la divulgation de tous les contenus protégés avec cette clé. De même, la compromission de la clé de chiffrement de contenu peut résulter en la divulgation du contenu chiffré qui y est associé.

Les mises en œuvre doivent protéger la clé privée de gestion de clé et la clé d'authentification de message. La compromission de la clé privée de gestion de clé permet de déguiser des données en données authentifiées. De même, la compromission de la clé d'authentification de message peut résulter en une modification indétectable du contenu authentifié.

La technique de gestion de clé employée pour distribuer les clés d'authentification de message doit elle-même fournir l'authentification de l'origine des données ; autrement, le contenu est livré avec la garantie d'intégrité provenant d'une source inconnue. Ni RSA [RFC2313], [RFC2437], ni le Diffie-Hellman statique-éphémère [RFC2631] ne fournissent l'authentification nécessaire de l'origine des données. Le Diffie-Hellman statique-statique [RFC2631] fournit bien l'authentification nécessaire de l'origine des données lorsque les clés publiques de l'origine et du receveur sont toutes deux liées aux identités appropriées dans les certificats X.509.

Lorsque plus de deux parties partagent la même clé d'authentification de message, l'authentification de l'origine des données n'est pas fournie. Toute partie qui connaît la clé d'authentification de message peut calculer un MAC valide; donc, le contenu pourrait avoir été généré par n'importe laquelle des parties.

Les mises en œuvre doivent générer au hasard les clés de chiffrement de contenu, les clés d'authentification de message, les vecteurs d'initialisation (IV), et le bourrage. Aussi, la génération de paires de clés publique/privée s'appuie sur des nombres aléatoires. L'utilisation de générateurs de nombres pseudo aléatoires (PRNG) inadéquats pour générer des clé de chiffrement peut résulter en une sécurité faible ou inexistante. Un attaquant peut trouver plus facile de reproduire l'environnement du PRNG qui a produit les clés, et chercher dans le petit ensemble de possibilités résultant, que de faire une recherche en force brute sur la totalité de l'espace de clés. La génération de nombres aléatoire de qualité est difficile. La [RFC4086] offre des lignes directrices importantes dans ce domaine.

Lorsque on utilise des algorithmes d'accord de clés ou des clés symétriques de chiffrement de clés précédemment distribuées, une clé de chiffrement de clé est utilisée pour chiffrer la clé de chiffrement de contenu. Si les algorithmes de chiffrement de clé et de chiffrement de contenu sont différents, la sécurité effective est déterminée par le plus faible des deux algorithmes. Si, par exemple, le contenu est chiffré avec Triple-DES en utilisant une clé de chiffrement de contenu Triple-DES à 168 bits, et si la clé de chiffrement de contenu est enveloppée avec RC2 en utilisant une clé de chiffrement de clé RC2 de 40 bits, ce ne sont pas plus de 40 bits de protection qui sont fournis. Une recherche triviale pour déterminer la valeur de la clé RC2 de 40 bits peut récupérer la clé Triple-DES, et ensuite la clé Triple-DES peut être utilisée pour déchiffrer le contenu. Donc, les mises en œuvre doivent s'assurer que les algorithmes de chiffrement de clé sont aussi forts ou plus forts que les algorithmes de chiffrement de contenu.

Les mises en œuvre devraient être conscientes que les algorithmes cryptographiques s'affaiblissent avec le temps. De nouvelles techniques de cryptanalyse sont développées et les performances de calcul s'améliorent, le facteur travail pour casser un algorithme cryptographique particulier en sera réduit. Donc, les mises en œuvre d'algorithmes cryptographiques devraient être modulaires, permettant que de nouveaux algorithmes soient directement insérés. C'est à dire que les mises en œuvre devraient être préparées à ce que l'ensemble d'algorithmes qui doivent être pris en charge change au fil du temps.

L'attribut de contreseing non signé comporte une signature numérique qui est calculée sur la valeur de la signature du contenu ; donc, le processus de contreseing n'a pas besoin de connaître le contenu signé original. Cette structure permet une amélioration de l'efficacité de la mise en œuvre ; cependant, cette structure peut aussi permettre le contreseing d'une valeur de signature inappropriée. Donc, les mises en œuvre qui effectuent le contreseing devraient, soit vérifier la valeur de la signature originale avant de la contresigner (cette vérification exige le traitement du contenu original) soit elles devraient effectuer le contreseing dans un contexte qui assure que seules les valeurs de signature appropriées sont contresignées.

15. Remerciements

Le présent document résulte des contributions de nombreux professionnels. Je remercie de leur dur labeur tous les membres du groupe de travail S/MIME de l'IETF. Je dois des remerciements tout particuliers à Rich Ankney, Simon Blake-Wilson, Tim Dean, Steve Dusse, Carl Ellison, Peter Gutmann, Bob Jueneman, Stephen Henson, Paul Hoffman, Scott Hollenbeck, Don Johnson, Burt Kaliski, John Linn, John Pawling, Blake Ramsdell, Francois Rousseau, Jim Schaad, Dave Solo, Paul Timmel, et Sean Turner pour leurs efforts et leur soutien.

Je remercie Tim Polk pour ses encouragements qui ont fait avancer cette spécification sur l'échelle de maturation des normes. De plus, je dois des remerciements à Jan Vilhuber pour sa relecture attentive qui a débouché sur l'errata à la RFC 1744.

Adresse de l'auteur

Russell Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA
mél : housley@vigilsec.com