rfc5652

Syntaxe de message cryptographique

Définitions

Ce document décris la syntaxe de message cryptographique (CMS). Cette syntaxe est utilisée pour signer, hasher, authentifier ou chiffrer numériquement du contenu de message arbitraire.

Le CMS décris une syntaxe d'encapsulation pour la protection des données. Il supporte les signatures et chiffrement numérique. La syntaxe permet plusieurs encapsulations ; un enveloppe d'encapsulation peut être imbriquée dans une autre. Également, une partie peut signer une donnée précédemment encapsulée. Il peut également des attributs arbitraires, telle que la date de signature.

Le CMS peut supporter divers architectures pour la gestion de clé basé sur les certificats, tes que celui définis par le groupe de travail PKIX.

Les valeurs CMS sont générées en utilisant ASN.1, utilisant l'encodage BER. Les valeurs sont typiquement représentées comme chaîne d'octets. Bien que de nombreux systèmes sont capable de transmettre des chaînes d'octet arbitraires de manière sûre, il est bien connu que les systèmes de messagerie électronique ne le sont pas. Ce document n'adresse pas les mécanismes pour encoder les chaînes d'octets pour une transmission sûre dans de tels environnements.

Évolution de CMS

Le CMS est dérivé de PKCS#7 version 1.5, qui est documenté dans la rfc 2315. PKCS#7 a été publié à l'origine comme note technique des laboratoires RSA en novembre 1993. IETF a depuis repris la responsabilité pour le développement et la maintenance du CMS. Aujourd'hui, de nombreux protocoles IETF utilisent CMS. Cette section décris les changement que l'IETF a fait de CMS dans chaque version publiée.

Changements depuis PKCS#7 version 1.5

la rfc 2630 [CMS1] était la première version de CMS. Quand cela est possible, la compatibilité avec PKCS#7 a été préservée ; cependant, des changements fait pour accueillir le transfert de l'attribut certificat version 1 et pour supporter la gestion de clé indépendamment de l'algorithme. PKCS#7 version 1.5 incluait le support seulement pour le transport des clé. la rfc 2630 ajoute le support pour l'accord de clé et les techniques de chiffrement de clé symétrique distribués précédemment.

Changements depuis la rfc 2630

La rfc 3369 [CMS2] remplace la rfc 2630 et la rfc 3211. La gestion de clé basé sur des mots de passe est inclue dans la spécification CMS et un mécanisme d'extension pour supporter de nouveau schémas de gestion de clé dans avoir à changer le CMS est spécifié. La compatibilité avec la rfc 2630 et la rfc3211 est préservée. Cependant, le transfert d'attribut de certificat version 2 est ajouté, et l'utilisation des attributs de certificat version 1 est déprécié.

Les signatures S/MIME v2, qui sont basées sur PKCS#7 v1.5, sont compatibles avec les signature S/MIME v3 et v3.1. Cependant, il y a quelques problème subtiles de compatibilité avec les signatures basées sur PKCS#7 v1.5.

Les algorithmes de chiffrement spécifiques ne sont pas discutés dans ce document, mais ont été discutés dans la rfc 2630. La discussion des algorithmes de chiffrement on été placés dans les document séparés.

Changements depuis la rfc 3369

La rfc 3852 [CMS3] remplace la rfc 3369. Comme discuté dans la précédente section, la rfc 3369 a introduit un mécanisme d'extension pour supporter de nouveaux schémas de gestion de clé. La rfc 3852 introduit un mécanisme d'extension similaire pour supporter des formats de certificats additionnels et les formats d'information de status de révocation.

Changement depuis la rfc 3852

Ce document remplace la rfc 3852. La raison principale pour la publication de ce document est de faire progresser le CMS avec la maturité des standards. Ce document inclus la clarification qui était originellement publiée dans la rfc 4853 concernant la manipulation correcte du type de contenu SignedData quand plus d'une signature numérique est présente.

Numéros de version

Chacune des structures de donnée majeur inclus un numéro de version comme premier élément de la structure. Les numéros de version. Les numéro de version sont prévus pour éviter les erreurs de décodage ASN.1. Certaines implémentations ne vérifient pas le numéro de version avant de tenter un décodage, et si une erreur est rencontrée, le numéro de version est vérifié. C'est une approche raisonnable.

La plupart des numéros de version initiales ont été assignés dans PKCS#7 v1.5. D'autre ont été assignés quand la structure a été créée initialement. Quand une structure est mise à jours, un numéro de version supérieur est assigné. Cependant, pour s'assurer d'une interopérabilité maximale, le numéro de version supérieur est uniquement utilisé quand une nouvelle syntaxe est employée.

Vue générale

Le CMS est suffisamment généraliste pour supporter de nombreux types de contenu différents. Ce document définis une protection de contenu, ContentInfo. ContentInfo encapsule un type de contenu identifié simple, et le type identifié peut fournir d'autres encapsulations. Ce document définis 6 types de contenu : donnée, donnée signée, donnée enveloppée, donnée hashée, donnée chiffrée et donnée authentifiée. Des types de contenu additionnels peuvent être définis en dehors de ce document.

Une implémentation qui se conforme à cette spécification doit implémenter le contenu de protection, ContentInfo, et doit implémenter les types de contenu donnée, donnée hashé donnée chiffrée, et donnée authentifiée. Les autres type de contenu peuvent être implémentés.

Comme philosophie de design générale, chaque type de contenu permet un traitement en une seule passe en utilisant l'encodage BER à longueur indéfinie. L'opération single-pass est spécifiquement utile si le contenu est grand, stocké sur cassettes, ou pipé depuis un autre processus. Une opération simple passe a un inconvénient important : il est difficile d'effectuer des opérations d'encodage en utilisant DER dans une simple passe vu que les longueurs des divers composants peuvent ne pas être connus à l'avance. Cependant, les attributs signés dans le type de contenu donnée signée et les attributs authentifiés dans le type de contenu donnée authentifiée doivent être transmis sous la forme DER pour s'assurer que le destinataire puisse vérifier un contenu qui contiendrait un ou plusieurs attributs non-reconnus. Les attributs signés et les attributs authentifiés sont les seuls types de donnée utilisées dans le CMS qui nécessitent un encodage DER.

Syntaxe générale

Les identifiants d'objet suivants identifient le type d'information de contenu :

```
id-ct-contentInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
smime(16) ct(1) 6 }
```

Le CMS associe un identifiant de type de contenu avec un contenu. La syntaxe doit avoir le type ASN.1 ContentInfo :

```
ContentInfo ::= SEQUENCE {
 contentType ContentType,
 content [0] EXPLICIT ANY DEFINED BY contentType }

 ContentType ::= OBJECT IDENTIFIER
```

contentType Indique le type de contenu associé. C'est un identifiant d'objet ; c'est une chaîne unique d'entiers assignés par une autorité qui définie le type de contenu.

content Est le contenu associé. Le type de contenu peut être déterminé de manière unique par le contentType. Les types de contenus pour donnée, donnée signée, donnée enveloppée, donnée hashée, donnée chiffrée et donnée chiffrée sont définies dans ce document. Si d'autres types de contenu sont définis dans d'autres documents, le type ASN.1 définis ne devrait pas être un type CHOICE.

Type de contenu donnée

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

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

Le type de contenu donnée est prévu pour référer à des chaînes d'octets arbitraires, tels que des fichiers texte ASCII ; l'interprétation est laissée à l'application. De telles chaînes n'ont pas besoin de structure interne (bien qu'ils peuvent avoir leur propre définition ASN.1 ou d'autre structure). S/MIME utilis id-data pour identifier le contenu encodé MIME. L'utilisation de cet identifiant de contenu est spécifié dans la rfc 2311 pour S/MIME v2, rfc 2633 pour S/MIME v3, et rfc 3851 pour S/MIME v3.1. Le type de contenu donnée est généralement encapsulée dans un type de contenu donnée signée, donnée enveloppée, donnée hashée, donnée chiffrée, ou donnée authentifiée.

Type de contenu donnée signée

Le type de contenu donnée signée consiste d'un contenu de n'importe quel type et 0 ou plusieurs valeurs de signatures. Plusieurs signataires en parallèle peuvent signer un type de contenu.

L'application typique du type de contenu donnée signée représente la signature numérique d'un signataire sur le contenu d'un type de contenu donnée. Une autre application typique est la diffusion de certificats et le listes de révocation de certificats. Le processus par lequel une donnée signée est construite est composé des étapes suivantes :

1. Pour chaque signataires, un hash est calculé sur le contenu avec un algorithme de hash spécifique au signataire. Si le signataire signe une information autre que le contenu, le hash du contenu et les autres informations sont hashées avec l'algorithme de hashage du signataire, et le résultat devient le hash.
2. Pour chaque signataire, le hash est signé numériquement en utilisant la clé privée du signataire.
3. Pour chaque signataire, la valeur de signature et d'autres informations spécifiques au signataire sont collectées dans une valeur SignerInfo. Les certificats et CRL pour chaque signataire, et ceux ne correspondant pas à un signataire, sont collectées dans cette étape.
4. Les algorithmes de hash et les valeurs SignerInfo pour tous les signataires sont collectés ensemble avec le contenu dans une valeur SignedData.

Un destinataire calcule de manière indépendante le hash. Ce hash est la clé publique du signataire sont utilisés pour vérifier la valeur de la signature. La clé publique du signataire est référencée d'un des 2 manières. Elle peut être référencée par un dn de fournisseur avec un numéro de série de fournisseur pour identifier de manière unique le certificat qui contient la clé publique. Alternativement, elle peut être référencéé par un identifiant de clé du sujet, qui reçoit les clé publiques certifiés et non-certifiés. Bien qui ce ne soit pas requis, le certificat du signataire peut être inclu dans le champ des certificates du SignedData.

Quand plus d'une signature est présente, la réussite de la validation d'une signature associée avec un signataire donné est généralement traitée comme une signature réussie dans le signataire. Cependant, il y a certaines application où d'autres règles sont nécessaires. Une application qui emploie une règle autre qu'une signature valide pour chaque signataire doit spécifier ces règles. Également, là où une simple correspondance de l'identifiant du signataire n'est pas suffisant pour déterminer si les signatures ont été générées par le même signataire, l'application doit décrire comment déterminer quelles signatures ont été générées par le même signataire. Le support de différentes communautés des bénéficiaires est la principale raison qui les signataires choisissent d'inclure plus d'une signature.

Par exemple, le type de contenu donnée signée peut inclure des signatures générées avec l'algorithme de signature RSA et avec l'algorithme de signature à courbe elliptique (ECDSA). Celà permet aux destinataires de vérifier la signature associée avec un algorithme ou l'autre.

Cette section est divisée en 6 parties. La première décris le type SignedData, la seconde décris EncapsulatedContentInfo, la troisième décris les information par signataire SignerInfo, la quatrième, cinquième et sixième décrivent le calcule du hash, génération de signature, et processus de vérification de signature.

Type SignedData

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

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

Le type de contenu donnée signée devrait avoir le type ASN.1 :

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

 DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

 SignerInfos ::= SET OF SignerInfo
```

version Est le numéro de version de syntaxe. La valeur appropriée dépend des certificats, eContentType et SignerInfo. a version doit être assignée comme suit :

```
IF ((certificates is present) AND
 (any certificates with a type of other are present)) OR
 ((crls is present) AND
 (any crls with a type of other are present))
THEN version MUST be 5
ELSE
 IF (certificates is present) AND
  (any version 2 attribute certificates are present)
 THEN version MUST be 4
 ELSE
  IF ((certificates is present) AND
   (any version 1 attribute certificates are present)) OR
```

```
   (any SignerInfo structures are version 3) OR
   (encapContentInfo eContentType is other than id-data)
  THEN version MUST be 3
  ELSE version MUST be 1
```

digestAlgorithm Est une collection d'identifiant d'algorithme de hash. Il peut y avoir plusieurs éléments dans la collection, incluant 0. Chaque élément identifie l'algorithme de hash, avec leur paramètres associés, utilisés par un ou plusieurs signataires. La collection est prévue pour lister les algorithmes de hash employées par tous les signataires, dans n'importe quel ordre, pour faciliter la vérification de signature en une passe. Les implémentations peuvent échouer la validation des signatures qui utilisent un algorithme qui n'est pas inclus dans ce set.

encapContentInfo Est le contenu signé, consistant d'un identifiant de type de contenu et le contenu lui-même.

certificates Est une collection de certificats. Il est prévu que ce jeu de certificat soit suffisant pour contenir les chemins de certification depuis un root reconnus vers tous les signataires dans le champ signerInfos. Il peut y avoir plus de certificats qui nécessaire, et il peut y avoir suffisamment de certificats pour contenir les chemins de certification depuis 2 ou plusieurs autorités de certification racine indépendante. Il peut y avoir moins de certificats qui nécessaire, s'il est prévu que les destinataires ont un moyen alternatif pour les obtenir. Le certificat du signataire peut être inclus. L'utilisation de la version 1 de l'attribut certificates est fortement découragée.

crls Est une collection d'information de status de révocation. Il est prévu qui la collection contienne les information suffisantes pour déterminer si les certificats dans le champ certificates sont valides, mais une telle correspondance n'est pas nécessaire. Les CRL sont la principale source d'information de status de révocation. Il peut y avoir plus de CRL que nécessaire, et il peut y avoir moins de crl qui nécessaire.

signerInfos Est une collection d'information par signataire. Il peut y avoir plusieurs éléments dans la collection, incluant 0. Quand la collection représente plus d'une signature, la réussite de la validation d'une signature d'un signataire donné doit être traitée comme une signature réussie par le signataire. Cependant, il y a certaines applications où d'autres règles sont nécessaire. Vu que chaque signataire peut employer une technique de signature différente, et de futures spécifications peut mettre à jours la syntaxe, toutes les implémentations doivent manipuler les versions non-implémentées de SignerInfo. De plus, toutes les implémentations doivent manipuler les algorithmes de signature non-implémentées quand elles sont rencontrées.

Type EncapsulatedContentInfo

Le contenu est représenté dans le type EncapsulatedContentInfo :

```
EncapsulatedContentInfo ::= SEQUENCE {
 eContentType ContentType,
 eContent [0] EXPLICIT OCTET STRING OPTIONAL }

 ContentType ::= OBJECT IDENTIFIER
```

eContentType Est un identifiant d'objet.

eContent Est le contenu lui-même, il n'a pas besoin d'être encodé DER.

L'omission optionnelle de eContent avec le champ EncapsulatedContentInfo rend possible la contruction de signatures externes. Dans le cas de signatures externes, le contenu à signer est absent de la valeur EncapsulatedContentInfo inclus dans le type de contenu donnée signée. Si la valeur eContent dans EncapsulatedContentInfo est absent, signatureValue est calculée et le eContentType est assignée comme si la valeur eContent était présente.

Dans le cas dégénéré où il n'y a pas de signataires, la valeur EncapsulatedContentInfo à signer est sans importance. Dans ce cas, le type de contenu avec la valeur EncapsulatedContentInfo à signer doit être id-data, et le champ content de la valeur EncapsulatedContentInfo doit être omise.

Compatibilité avec PKCS#7

Cette section contient un mot d'alerte pour les implémenteurs qui souhaites supporter les type de contenu SignedData CMS et PKCS#7. CMS et PKCS#7 identifient le type de contenu encapsulé avec un identifiant d'objet, mais le type ASN.1 du contenu lui-même est variable dans le type de contenu SignedData PKCS#7.

PKCS#7 définis le contenu :
content [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL
Le CMS définis eContent :
eContent [0] EXPLICIT OCTET STRING OPTIONAL
La définition de CMS est plus facile à utiliser dans la plupart des applications, et est compatible avec S/MIME v2 et v3. les messages signés S/MIME utilisant CMS et PKCS#7 sont compatibles parce que les format de message signés identique sont spécifiés dans les rfc 2311, 2633 et 3851. S/MIME v2 encapsule le contenu MIME dans un type Data (un OCTET STRING) dans le contenu contentInfo de SignedData, et S/MIME v2 le porte dans le eContent encapContentInfo SignedData. Cependant, dans S/MIME v2, v3 et v3.1, le contenu MIME est placé dans un OCTET STRING et le hash est calculé sur les portions identiques de contenu.

Il y a des incompatibilités entre les type SignedData de CMS et PKCS#7 qui le contenu encapsulé n'est pas formaté en utilisant le type Data. Par exemple, qui un reçu signé rfc3634 est encapsulé dans le type SignedData CMS, le Receipt SEQUENCE est encodé dans le SignedData encapContentInfo eContent OCTET STRING et le hash est calculé en utilisant tous l'encodage Receipt SEQUENCE (incluant le tag, longueur et octets de valeurs). Cependant, si un reçu signé rfc2634 est encapsulé dans le type SignedData PKCS#7, alors le Receipt SEQUENCE est encodé DER dans le contenu SignedData contentInfo (une SEQUENCE, par un OCTET STRING). Cependant, le hash est calculé en utilisant seulement les octets de valeur de l'encodage de Receipt SEQUENCE.

La stratégie suivante peut être utilisée pour la compatibilité avec PKCS#7 en traitant les type de contenu SignedData. Si l'implémentation n'est pas capable de décoder l'ASN.1 du type SignedData en utilisant la syntaxe CMS SignedData encapContentInfo eContent OCTET STRING, alors l'implémentation peut tenter de décoder le type SignedData en utilisant une syntaxe de contenu PKCS#7 SignedData contentInfo et calculer le hash.

La stratégie suivante peut être utilisé pour la compatibilité avec PKCS#7 en créant un type de contenu SignedData dans lequel le contenu encapsulé n'est pas formaté en utilisant le type Data. Les implémentations peuvent examiner la valeur de eContentType, et ainsi ajuster l'encodage DER attendu de eContent basé sur la valeur de l'identifiant d'objet. Par exemple pour supporter MSAC, les informations suivantes peuvent être inclus :

```
 eContentType Object Identifier is set to { 1 3 6 1 4 1 311 2 1 4 }

 eContent contains DER-encoded Authenticode signing information
```

Type SignerInfo

Les information par signataire sont représentés dans le type SignerInfo :

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

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

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute
```

```
UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
 attrType OBJECT IDENTIFIER,
 attrValues SET OF AttributeValue }

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING
```

version Est le numéro de version de syntaxe. Si SignerIdentifier est le choix issuerAndSerialNumber, la version doit être 1. Si SignerIdentifier est sibjectKeyIdentifier, la version doit être 3.

sid Spécifie le certificat du signataire (et donc sa clé publique). La clé publique du signataire est nécessaire au destinataire pour vérifier la signature.

SignerIdentifier fournis 2 alternatives pour spécifier la clé publique du signataire. L'alternative issuerAndSerialNumber identifie le certificat du signataire par le dn du fournisseur et le numéro de série du certificat ; le subjectKeyIdentifier identifie le certificat du signataire par un identifiant de clé. Quand un certificat X.509 est référencé, l'identifiant de clé matche la valeur de l'extention subjectKeyIdentifier. Quand d'autres formats de certificats sont référencés, les documents qui spécifient le format de certificat et leur utilisation avec le CMS doivent inclure les détails sur le matching d'identifiant de clé du champ certificat approprié. Les implémentations doivent supporter la réception des formes issuerAndSerialNumber et subjectKeyIdentifier.

En générant un SignerIdentifier les implémentations peuvent supporter une des formes (soit issuerAndSerialNumber soit subjectKeyIdentifier) et toujours l'utiliser, ou les implémentations peuvent arbitrairement mixer les 2 formes. Cependant, subjectKeyIdentifier doit être utilisé pour référer à une clé publique contenue dans un certificat non-X.509.

digestAlgorithm Identifie l'algorithme de hash, et ses paramètres associés, utilisés par le signataire. Le hash est calculé sur soit le contenu à signer soit le contenu et les attributs signés en utilisant le processus décris plus bas. l'algorithme de hash devrait être listé dans le champ digestAlgorithms du SignerData associé. Les implémentations peuvent échouer si ce n'est pas le cas.

signedAttrs Est une collection d'attributs qui sont signés. Le champ est optionnel, mais il doit être présent si le type de contenu de la valeur EncapsulatedContentInfo à signer n'est pas id-data. SignedAttributes doit être encodé DER, même si le reste de la structure set encodée BER. Les types d'attributs utiles, tels qu'une date de signature, sont définis plus loin dans ce document. Si le champ est présent, il doit contenir, au minimum, les 2 attributs suivants :

content-type ayant comme valeur le type de contenu de la valeur EncapsulatedContentInfo à signer. Cependant, cet attribut ne doit pas être utilisé comme partie d'un attribut contersignatures non-signé

message-digest ayant comme valeur le hash du contenu.

signatureAlgorithm identifie l'algorithme de signature, et ses paramètres associés, utilisé par le signataire pour générer la signature numérique.

signature Est le résultat de la génération de la signature, en utilisant le hash et la clé privée du signataire.

unsignedAttrs Est une collection d'attributs qui ne sont pas signés. Ce champ est optionnel.

Les champs de type SignedAttributes et UnsignedAttributes ont la signification suivante :

attrType Indique le type d'attribut, c'est un identifiant d'objet.

attrValues Est un jeu de valeurs pour cet attribut.

Processus de calcul du hash

Le processus de calcul du hash calcule un hash sur soit le contenu à signer, soit le contenu et les attributs signés. Dans tous les cas, l'entrée initiale du processus est la valeur du contenu encapsulé à signer. Spécifiquement, l'entrée initiale est la chaîne d'octets eContent de encapContentInfo auquel le processus de signature est appliqué. Seul les octets comprenant la valeur de la chaîne d'octets eContent sont entrés dans l'algorithme de hashage, et non le tag ou les octets de longueur.

Le résultat du processus de calcul du hash dépend de la présence ou non du champ signedAttrs. Quand le champ est absent, le résultat est simplement le hash du contenu. Quand le champ est présent, cependant, le résultat est le hash de l'encodage DER complet de la valeur SignedAttrs contenu dans le champ signedAttrs. Vu que la valeur SignedAttrs, quand présent, doit contenir les attributs de type de contenu et de hash, ces valeurs sont indirectement incluses dans le résultat. L'attribut content-type ne doit pas être inclus dans un attribut non signé countersignature. Un encodage séparé du champ signedAttrs est effectué pour le calcul du hash. Le IMPLICIT [0] tag dans signedAttrs n'est pas utilisé pour l'encodage DER, et EXPLICIT SET OF tag est utilisé, et doit donc être inclus dans le calcul du hash avec les octets de longueur et de contenu de la valeur SignedAttributes.

Quand le champ signedAttrs est absent, seul les octets compris dans la valeur de la chaîne d'octets eContent de encapContentInfo du SignedData sont entrés dans le calcul du hash. Cela a l'avantage que la longueur du contenu à signer n'a pas besoin d'être connus à l'avance.

Bien que le les octets de tag et de longueur de la chaîne d'octets eContent de encapContentInfo ne sont pas inclus dans le calcul du hash, ils sont protégés par d'autres moyens. Les octets de longueur sont protégés par la nature de l'algorithme.

Processus de génération de signature

L'entrée du processus de génération de signature inclus le résultat du processus de calcul du hash et la clé privée du signataire. Les détails de la génération de la signature dépend de l'algorithme de signature employé. L'identifiant d'objet, avec ses paramètres, qui spécifient l'algorithme de signature employé par le signataire sont placés dans le champ signatureAlgoritm. La valeur de la signature générée par le signataire doit être encodé en chaîne d'octets et placés dans le champ signature.

Processus de vérification de signature

L'entrée du processus de vérification de signature inclus le résultat du processus de calcul du hash et la clé publique du signataire. Le destinataire peut obtenir la bonne clé publique pour le signataire par tous moyens, mais la méthode préférée est depuis un certificat obtenu depuis le champ certificates de SignedData. La sélection et la validation de la clé publique du signataire peut être basée sur la validation du chemin de certification aussi bien que d'autres contextes externes, mais c'est au-delà du scope de ce document. Les détails de la vérification de la signature dépendent de l'algorithme de signature employé.

Le destinataire ne doit pas s'appuyer sur les valeurs de hash calculés par l'auteur. Si le signerInfo de SignedData inclus un SignedAttributes, alors le hash doit être calculé comme décris plus haut. Pour que la signature soit valide, la valeur du hash calculée par le destinataire doit être la même que la valeur de l'attribut messageDigest inclus dans le signedAttributes du signerInfo du SignedData.

Si signerInfo du SignedData inclus signedAttributes, alors la valeur de l'attribut content-type doit matcher la valeur eContentType de encapContentInfo du signedData.

Type de contenu Enveloped-data

Le type de contenu donnée enveloppée consiste d'un contenu chiffré et de clé de chiffrement de contenu chiffrés pour un ou plusieurs destinataires. La combinaison du contenu chiffré et un clé de chiffrement de contenu chiffré pour un destinataire est une enveloppe numérique pour ce destinataire. Tout type de contenu peut être enveloppé pour un nombre arbitraire de destinataires utilisant tout type de technique de gestion de clé supportés pour chaque destinataire.

L'application typique du type de contenu donnée enveloppée représente une ou plusieurs enveloppes numérique de destinataires sur le contenu des type de de contenu donnée et donnée signée. Enveloped-data est construit par les étapes suivantes :

1. Une clé de chiffrement pour un algorithme de chiffrement de contenu est généré aléatoirement.

2. La clé de chiffrement de contenu est chiffrée pour chaque destinataire. Les détails de ce chiffrement dépend de l'algorithme de gestion de clé utilisé, mais 4 techniques générales sont supportées :

   transport de clé La clé de chiffrement de contenu est chiffrée dans la clé publique clé publique du destinataire.

   agrément de clé La clé publique du destinataire est la clé privée de l'émetteur sont utilisé pour générer une clé symétrique, puis la clé de chiffrement de contenu est chiffré dans la clé symétrique.

   Clé de chiffrement symétrique La clé de chiffrement de contenu est chiffré dans un clé de chiffrement de clé symétrique distribuée précédemment.

   mot de passes La clé de chiffrement de contenu est chiffrée dans une clé de chiffrement de contenu qui est dérivée d'un mot de passe ou autre valeur de secret partagée.

3. Pour chaque destinataire, la clé de chiffrement de contenu chiffré et d'autres informations spécifiques au destinataire sont collectés dans une valeur RecipientInfo.
4. Le contenu est chiffré avec la clé de chiffrement de contenu. Le chiffrement de contenu peut nécessiter qui le contenu soit padded à un multiple d'une taille de block.
5. Les valeurs RecipientInfo pour tous les destinataires sont collectés ensemble avec le contenu chiffré pour former une valeur enveloppée.

Un destinataire ouvre l'enveloppe numérique en déchiffrant celle qui contient les clé de chiffrement de contenu chiffrés puis déchiffre le contenu chiffré avec la clé de chiffrement de contenu récupéré.

Cette section est divisée en 4 parties. La première partie décris le type EnvelopedData, la seconde partie décris le type d'information par destinataire RecipientInfo, et les troisième et quatrième partie décrivent le chiffrement de contenu et les processus de chiffrement de clé.

Type EnvelopedData

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

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

Le type donnée enveloppée devrait avoir le type ASN.1 suivant :

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

OriginatorInfo ::= SEQUENCE {
 certs [0] IMPLICIT CertificateSet OPTIONAL,
 crls [1] IMPLICIT RevocationInfoChoices OPTIONAL }

RecipientInfos ::= SET SIZE (1..MAX) OF RecipientInfo

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

EncryptedContent ::= OCTET STRING

UnprotectedAttributes ::= SET SIZE (1..MAX) OF Attribute
```

version Est le numéro de version de la syntaxe. La valeur appropriée dépend de originatorInfo, RecipientInfo, et unprotectedAttrs. La version doit être assignée comme suit :

```
IF (originatorInfo is present) AND
 ((any certificates with a type of other are present) OR
 (any crls with a type of other are present))
THEN version is 4
ELSE
 IF ((originatorInfo is present) AND
  (any version 2 attribute certificates are present)) OR
  (any RecipientInfo structures include pwri) OR
  (any RecipientInfo structures include ori)
 THEN version is 3
 ELSE
  IF (originatorInfo is absent) AND
   (unprotectedAttrs is absent) AND
   (all RecipientInfo structures are version 0)
  THEN version is 0
  ELSE version is 2
```

originatorInfo Fournis optionnellement des informations sur l'auteur. Il est présent seulement s'il est requis par l'algorithme de gestion des clés. Il peut contenir des certificats et des crls :

certs est une collection de certificats. certs peut contenir les certificats d'expéditeur associés avec de nombreux algorithmes de gestion de clés. certs peut également contenir des certificats d'attribut associés avec l'expéditeur. Les certificats contenus dans certs sont prévus pour être suffisants pour tous les destinataires pour construire des chemin de validation depuis un root reconnu. Cependant, certs peut contenir plus de certificats que nécessaire, et il peut y avoir les certificats nécessaire pour créer les chemins de validation de 2 ou plusieurs autorité de certification racine. Alternativement, il est prévu que les destinataires aient un moyen alternatif pour obtenir les certificats nécessaires.

crls est une collection de CRL. Il est prévu que ce jeu contienne les informations suffisantes pour déterminer si les certificats dans le champ certs sont valides ou non. Il peut y avoir plus de CRL que nécessaire, et il peut y avoir moins de crl que nécessaire.

recipientInfos est une collection d'information par destinataire, il doit y avoir au moins un élément dans la collection.

encryptedContentInfo Est l'information de contenu chiffré

unprotectedAttrs est une collection d'attributs qui ne sont pas chiffrés. Le champ est optionnel.

EncryptedContentInfo Les champs ont la signification suivante :

contentType Indique le type de contenu

contentEncryptionAlgorithm identifie l'algorithme de chiffrement de contenu, et ses paramètres associés, utilisés pour chiffrer le contenu. Le même algorithme de chiffrement de contenu et la clé de chiffrement de contenu sont utilisé pour tous les destinataires.

Type RecipientInfo

Les informations par destinataire sont représentés dans le type RecipientInfo. RecipientInfo a un format différent pour chacune des techniques de gestion de clés supportés. Tout type de technique de gestion de clé peut être utilisé pour chaque destinataire de même contenu chiffré. Dans tous les cas, la clé de chiffrement de contenu chiffrée est transféréeà un ou plusieurs destinataires.

Vu que toutes les implémentations ne vont pas supporter tous les algorithmes de gestion de clés possibles, toutes les implémentations doivent gérer les algorithmes non-implémentés quand ils sont rencontrés. Par exemple, si un destinataire reçoit une clé de chiffrement de contenu chiffrée avec leur clé publique RSA en utilisant RSA-OAEP et que l'implémentation ne supporte que RSA PKCS#1 v1.5, alors l'échec doit être implémentée.

Les implémentations doit supporter le transport de clé, l'agrément de clé, et les clé de chiffrement de clé symétriques distribués précédemment, comme représentés par ktri, kari, et kekri, respectivement. Les implémentations peuvent supporter la gestion de clé basé sur mot de passe comme représenté par pwri. Les implémentations peuvent supporter d'autres technique de gestion de clé comme représenté par ori. Vu que chaque destinataire peut employer une technique de gestion de clé différente et que les spécifications futures peuvent définir des technique supplémentaires, toutes les implémentation doit manipuler les alternatives non-implémentées, les versions non-implémentées et les ori inconnues.

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

 EncryptedKey ::= OCTET STRING
```

Type KeyTransRecipientInfo

Les informations par destinataire utilisant le transport de clé sont représentés dans le type KeyTransRecipientInfo. Chaque instance de KeyTransRecipientInfo transfert la clé de chiffrement de contenu à un destinataire.

```
KeyTransRecipientInfo ::= SEQUENCE {
 version CMSVersion, - always set to 0 or 2
 rid RecipientIdentifier,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }

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

version Est le numéro de version de la syntaxe. Si RecipientIdentifier est issuerAndSerialNumber, la version doit être 0. Si c'est subjectKeyIdentifier, la version doit être 2.

rid Spécifie le certificat ou la clé du destinataire qui a été utilisé par l'émetteur pour protéger la clé de chiffrement de contenu. RecipientIdentifier fournis 2 alternatives pour spécifier le certificat du destinataire, et donc la clé publique du destinataire. Le certificat du destinataire doit contenir une clé publique de transport de clé. Donc, un certification X.509v3 d'un destinataire qui contient un extension d'utilisation de clé doit avoir le bin keyEncipherment. L'alternative issuerAndSerialNumber identifie le certificat du destinataire par le dn du fournisseur et le numéro de série du certificat ; le sibjectKeyIdentifier identifie le certificat du destinataire par un identifiant de clé.

Quand un certificat X.509 est référencé, l'identifiant de clé matche la valeur de l'extension subjectKeyIdentifier X.509. Quand d'autres formats de certificat sont référencés, les documents qui spécifient le format du certificat et son utilisation avec le CMS doit inclure les détails sur la correspondance le l'identifiant de clé. Pour le traitement du destinataire, les implémentations doivent supporter ces alternatives pour spécifier le certificat du destinataire. Pour l'émetteur, les implémentations doivent supporter au moins un de ces alternatives.

keyEncryptionAlgorithm Identifie l'algorithme de chiffrement de clé et ses paramètres associés, utilisés pour chiffrer la clé de chiffrement de contenu pour le destinataire.

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

Type KeyAgreeRecipientInfo

Les informations de destinataire utilisant l'accord de clé est représenté dans le type KeyAgreeRecipientInfo. Chaque instance de KeyAgreeRecipientInfo va transférer la clé de chiffrement de contenu à un ou plusieurs destinataires qui utilisent le même algorithme d'accord de clé et les paramètres de domaine pour cet algorithme.

```
KeyAgreeRecipientInfo ::= SEQUENCE {
 version CMSVersion, - always set to 3
 originator [0] EXPLICIT OriginatorIdentifierOrKey,
 ukm [1] EXPLICIT UserKeyingMaterial OPTIONAL,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
```

```
 recipientEncryptedKeys RecipientEncryptedKeys }

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

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

RecipientEncryptedKeys ::= SEQUENCE OF RecipientEncryptedKey

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

KeyAgreeRecipientIdentifier ::= CHOICE {
 issuerAndSerialNumber IssuerAndSerialNumber,
 rKeyId [0] IMPLICIT RecipientKeyIdentifier }
RecipientKeyIdentifier ::= SEQUENCE {
 subjectKeyIdentifier SubjectKeyIdentifier,
 date GeneralizedTime OPTIONAL,
 other OtherKeyAttribute OPTIONAL }

SubjectKeyIdentifier ::= OCTET STRING
```

version Est le numéro de version de la syntaxe. Doit toujours être 3.

originator Est un choix avec 3 alternatives spécifiant la clé publique de l'accord de clé de l'émetteur. L'émetteur utilise la clé privée correspondant et la clé publique du destinataire pour générer une clé paire. La clé de chiffrement de contenu est chiffrée dans la clé paire. L'alternative issuerAndSerialNumber identifie le certificat de l'émetteur, et donc sa clé publique, par dn de fournisseur et numéro de série. l'alternative subjectKeyIdentifier identifie le certificat de l'émetteur par identifiant de clé.

Quand un certificat X.509 est référencé, l'identifiant de clé matche la valeur de l'extension X.509 subjectKeyIdentifier. Quand d'autres format de certificat sont référencés, les documents qui spécifient le format de certificat et son utilisation avec le CMS doit inclure les détails sur le matche d'identifiant de clé. L'alternative originatorKey inclus l'identifiant d'algorithme et la clé publique d'accord de clé de l'émetteur. Cette alternative permet à l'auteur l' anonymité vu que la clé publique n'est pas certifiée. Les implémentations doivent suppporter les 3 alternatives pour spécifier la clé publique de l'émetteur.

ukm est optionnel. Avec certains algorithmes d'accord de clé, l'émetteur fournis un User Keying Material (UKM) pour s'assurer qu'une clé différente est générée chaque fois que les même 2 parties génèrent une clé paire. Les implémentations doivent accepter une séquence KeyAgreeRecipientInfo qui inclus un champ ukm. Les implémentations qui ne supportent par les algorithmes d'accord de clé qui utilisent UKM doivent manipuler correctement la présence des UKM.

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

recipientEncryptedKeys Inclus un identifiant de destinataire et une clé chiffrée pour un ou plusieurs destinataires. le KeyAgreeRecipientIdentifier est un choix avec 2 alternatives spécifiant le certificat du destinataire, et donc sa clé publique, qui a été utilisée par l'émetteur pour générer un clé de chiffrement de clé paire. Le certificat du destinataire doit contenir une clé publique d'accord de clé. Donc, un certificat X.509 v3 d'un destinataire qui contient une utilisation de clé étendue doit avoir le bit keyAgreement. La clé de chiffrement de contenu est chiffrée dans la clé de chiffrement de clé paire. L'alternative issuerAndSerialNumber identifie le certificat du destinataire par dn du fournisseur et numéro de série. encryptedKey est le résultat du chiffrement de la clé de chiffrement de contenu dans la clé de chiffrement de clé paire générée en utilisa l'algorithme d'accord de clé. Les implémentations doivent supporter ces alternatives en spécifiant le certificat du destinataire.

subjectKeyIdentifier Identifie le certificat du destinataire par identifiant de clé.

date est optionnel. Quand présent, la date spécifie celle du précédent UKM distribué précédemment qui a été utilisé par l'émetteur.

other est optionnel. Quand présent, ce champ contient des informations additionnels utilisé par le destinataire pour localiser l'UKM utilisé par l'émetteur.

Type KEKRecipientInfo

Les informations de déstinataire utilisant les clés symétriques précédemment distribuées sont représentés dans le type KEKRecipientInfo. Chaque instance de KEKRecipientInfo va transférer la clé de chiffrement de contenu à un ou plusieurs destinataires qui ont la clé de chiffrement de clé distribuée précédemment.

```
KEKRecipientInfo ::= SEQUENCE {
 version CMSVersion, - always set to 4
 kekid KEKIdentifier,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }

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

version est le numéro de version de la syntaxe. Doit être 4.

kekid spécifie un clé de chiffrement de clé symétrique qui a été distribué précédemment à l'émetteur en un ou plusieurs destinataires.

keyEncryptionAlgorithm Identifie l'algorithme de chiffrement de clé et ses 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 dans la clé de chiffrement de clé.

keyIdentifier identifie la clé de chiffrement de clé qui a été précédemment distribuée à l'émetteur et un ou plusieurs destinataires.

date optionnel. Quand présent, la date spécifie un clé de chiffrement de clé simple depuis un jeu qui a été distribué précédemment.

other optionnel. Quand présent, ce champ contient des informations optionnelles utilisées par le destinataire pour déterminer la clé de chiffrement de clé utilisée par l'émetteur.

Type PasswordRecipientinfo

Les information de destinataire utilisant un mot de passe ou une valeur secrète partagée est représentée dans le type PasswordRecipientinfo. Chaque instance de PasswordRecipientinfo va transférer la clé de chiffrement de contenu à un ou plusieurs destinataires qui possède un mot de passe ou une valeur secrète partagée. Le type PasswordRecipientinfo est définie dans la rfc3211 :

```
PasswordRecipientInfo ::= SEQUENCE {
 version CMSVersion, - Always set to 0
 keyDerivationAlgorithm [0] KeyDerivationAlgorithmIdentifier
  OPTIONAL,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }
```

version est le numéro de version de la syntaxe. Doit être 0.

keyDerivationAlgorithm identifie l'algorithme de dérivation de clé et ses paramètres associés, utilisé pour dériver la clé de chiffrement de clé du mot de passe ou de la valeur secrète partagée. Si ce champ est absent, la clé de chiffrement de clé est fournie depuis une source externe, par example, un jeton cryptographique physique tel qu'une carte à puce.

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

Type OtherRecipientinfo

Les information de destinataire pour les techniques de gestion de clé additionnels sont représentés dans le type OtherRecipientInfo. Le type OtherRecipientInfo permet aux technique de gestion de clé au delà des type précédemment définis d'être spécifiés dans de futures documents. Un identifiant d'objet identifie de manière unique de telles techniques de gestion de clé.

```
OtherRecipientInfo ::= SEQUENCE {
 oriType OBJECT IDENTIFIER,
 oriValue ANY DEFINED BY oriType }
```

oriType identifie la technique de gestion de clé

oriValue contient les éléments de données du protocole nécessaire pour un destinataire utilisant la technique de gestion de clé identifiée.

Processus de chiffrement de contenu

La clé de chiffrement de contenu pour l'algorithme de chiffrement de contenu désiré est généré aléatoirement. La donnée à protéger est padded comme décris plus bas, puis cette donnée est chiffrée en utilisant la clé de chiffrement de contenu. L'opération de chiffrement mappe une chaîne d'octets arbitraire (la donnée) en une autre chaîne d'octets (le texte chiffré) sous le contrôle d'une clé de chiffrement de contenu. La donnée chiffrée est inclue dans encryptedContent de encryptedContentInfo du EnvelopedData.

Certains algorithmes de chiffrement de contenu assument que la longueur d'entrée est un multiple de k octets, où k est supérieur à 1. Pour de tels algorithmes, l'entrée devrait être complétée à la fin avec des octets k-(lth mod k) ayant pour valeur (k-lth mod k), où lth est la longueur de l'entrée. En d'autres termes, l'entrée est complétées à la fin avec une des chaînes suivantes :

```
_____01 - if lth mod k = k-1
_____02 02 - if lth mod k = k-2
_____.
_____.
_____.
k k ... k k - if lth mod k = 0
```

Le padding peut être supprimé de manière non ambigu vu que toute l'entrée est paddée, incluant les valeurs d'entrée qui sont déjà un multiple de la taille de block, et aucune chaîne de padding n'est un suffix d'un autre. Cette méthode de padding est définie si et seulement si k est inférieur à 256.

Processus de chiffrement de clé

L'entrée du processus de chiffrement de clé – la valeur fournie à l'algorithme de chiffrement de clé du destinataire – est simplement la valeur de la clé de chiffrement de contenu.

N'importe quelle technique de gestion de clé mentionnées dans ce document peuvent être utilisées pour chaque destinataire de même contenu chiffré.

Type de contenu Digested-Data

Le type de contenu donnée hashée consiste d'un contenu de n'importe quel type et un hash du contenu. Typiquement, le type de contenu donnée hashée est utilisé pour fournir une intégrité de contenu, et le résultat devient généralement un entrée au type de contenu donnée enveloppée. Les étapes suivantes construisent une donnée hashée :

1. Un hash est calculé sur le contenu avec un algorithme de hashage.
2. L'algorithme de hashage et le hash sont collectés ensemble avec le contenu dans une valeur DigestedData.

Un destinataire vérifie le hash en comparant le hash à un hash calculé indépendamment.

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

```
id-digestedData OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 5 }
```

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

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

Digest ::= OCTET STRING
```

version est le numéro de version de la syntaxe. Si le type de contenu incapsulé est id-data, la valeur doit être 0, 2 sinon.

digestAlgorithm identifie l'algorithme de hashage et ses paramètres associés.

encapContentInfo est le contenu qui est hashé

digest est le résultat du processus de hashage.

L'ordre des champs rend possible le traitement d'un valeur DigestedData en une seule passe.

Type de contenu Encrypted-data

Le type de contenu donnée chiffrée consiste d'un contenu de n'importe quel type, chiffré. À la différence du type de contenu donnée enveloppée, le type de contenu donnée chiffrée n'a ni destinataire, ni clé de chiffrement de contenu. Les clé doivent être gérées par un autre moyen.

L'application typique du type de contenu chiffré est de chiffré le contenu du type de contenu donnée pour un stockage local, où la clé de chiffrement est dérivée d'un mot de passe.

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

```
id-encryptedData OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 6 }
```

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

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

version est le numéro de version de la syntaxe. si unprotectedAttrs est présent, la version doit être 2, 0 sinon.

encryptedContentInfo est le contenu chiffré

unprotectedAttrs est une collection d'attributs qui ne sont pas chiffrés. Ce champ est optionnel.

Type de contenu Authenticated-data

Le type de contenu donnée authentifiée consiste d'un contenu de n'importe quel type, un MAC, et de clés d'authentification chiffrés pour un ou plusieurs destinataires. La combinaison du MAC et d'une clé d'authentification chiffrée pour un destinataire est nécessaire pour que ce destinataire puisse vérifier l'intégrité du contenu. Tout type de contenu peut être protégé pour l'intégrité pour plusieurs destinataires.

Le processus par lequel un donnée authentifiée est construite est faite des étapes suivantes :

1. Une clé d'authentification de message pour un algorithme d'authentification de message particulier est généré aléatoirement.
2. La clé d'authentification de message est chiffrée pour chaque destinataire. Les détails de ce chiffrement dépend de l'algorithme de gestion de clé utilisé.
3. Pour chaque destinataire, la clé d'authentification de message chiffrée et les autres information spécifiques au destinataire sont collectés dans une valeur RecipientInfo.
4. Un utilisant la clé d'authentification de message, l'auteur calcul une valeur MAC sur le contenu. Si l'auteur authentifie une information en plus du contenu, un hash est calculé sur le contenu, le hash du contenu est les autres informations sont authentifiées en utilisant la clé d'authentification de message, et le résultat devient la valeur MAC.

Type AuthenticatedData

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

```
id-ct-authData OBJECT IDENTIFIER ::= { 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 devrait avoir le type ASN.1 :

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

AuthAttributes ::= SET SIZE (1..MAX) OF Attribute

UnauthAttributes ::= SET SIZE (1..MAX) OF Attribute

MessageAuthenticationCode ::= OCTET STRING
```

version est le numéro de version de la syntaxe. La version doit êtré assignée comme suit :

```
IF (originatorInfo is present) AND
 ((any certificates with a type of other are present) OR
 (any crls with a type of other are present))
THEN version is 3
ELSE
 IF ((originatorInfo is present) AND
  (any version 2 attribute certificates are present))
 THEN version is 1
 ELSE version is 0
```

originatorInfo fournis optionnellement des informations sur l'auteur. Il est présent seulement s'il est requis par l'algorithme de gestion de clé. Il peut contenir des certificats, des attributs de certificat, et des CRLs.

recipientInfos Est une collection d'information par destinataire. Il doit y avoir au moins un élément dans la collection.

macAlgorithm est un identifiant d'algorithme MAC. Il identifie l'algorithme MAC et ses paramètres associés, utilisé par l'auteur.

digestAlgorithm identifie l'algorithme de hashage et ses paramètres associés, utilisé pour calculer un hash sur le contenu encapsulé si des attributs authentifiés sont présents. Si ce champ est présent, authAttrs doit l'être également.

authAttrs est une collection d'attributs authentifiés. La structure AuthAttributes doit être encodé DER, même si le reste du la structure est encodée BER. Si cet attribut est présent, il doit contenir au minimum ces 2 attributs :

content-type ayant comme valeur le type de contenu de la valeur EncapsulatedContentInfo qui est authentifée.

message-digest ayant comme valeur le hash du contenu.

mac est le message authentication code

unauthAttrs est une collection d'attributs non authentifiés. Ce champ est optionnel.

Génération du MAC

Le processus de calcul du MAC calcule un code d'authentification de message sur soit le contenu à authentifier soit un hash du contenu à authentifier avec les attributs authentifié de l'auteur.

Si le champ authAttrs est absent, l'entrée du processus de calcul MAC est la valeur de la chaîne d'octets de eContent dans encapContentInfo. Seul les octets comprenant la valeur de eContent sont entrées dans l'algorithme MAC ; le tag et les octets de longueur sont omis. Cela a l'avantage que la longueur du contenu à authentifier n'a pas besoin d'être connu à l'avance.

Si le champ authAttrs est présent, les attributs content-type et message-digest doivent être inclus, et l'entrée dans le processus de calcul du mac est l'encodé DER de authAttrs. Un encodage séparé du champ authAttrs est effectué pour un calcul de hash. Le tag implicite dans le champ authAttrs n'est pas utilisé pour l'encodage DER, un jeu explicite de tag est utilisé à la place. ce qui signifie que l'encodé DER du jeu de tag, au lieu du tag implicite, doit être inclus dans le calcul du hash avec les octets de longueur et de contenu de la valeur authAttrs.

Le processus de calcule du hash calcul un hash sur le contenu à authentifier. L'entrée initiale du processus de calcul du hash est la valeur du contenu encapsulé à authentifier. Spécifiquement, l'entrée est la chaîne d'octets de eContent de encapContentInfo pour lequel le processus d'authentification est appliqué. Seul les octets comprenant la valeur de la chaîne d'octets eContent de encapContentInfo sont entrés dans l'algorithme de hashage, pas le tag ni les octets de longueur. Cela a l'avantage que la longueur du contenu n'a pas besoin d'être connu à l'avance. Bien que le tag et les octets de longueur ne sont pas inclus dans le calcul du hash, il sont protégés par d'autres moyens. Les octets de longueur sont protégés par la nature de l'algorithme de hashage.

L'entrée du processus de calcul MAC inclus la donnée d'entrée MAC définie précédemment, et une clé d'authentification transportée dans une structure recipientInfo. Les détails du calcul MAC dépend de l'algorithme utilisé. L'identifiant d'objet et ses paramètres, qui spécifie l'algorithme employé par l'auteur est placé dans le champ macAlgorithm. La valeur MAC générée par l'auteur est encodé en chaîne d'octet dans le champ mac.

Vérification du MAC

L'entrée du processus de vérification du MAC inclus les donnée d'entrée (déterminées sur la présence ou non du champ authAttrs), et la clé d'authentification dans recipientInfo. Les détails du processus de vérification du MAC dépend de l'algorithme MAC employé.

Le destinataire ne doit pas se fier aux valeurs MAC et hash de l'auteur. Pour que l'authentification réussisse, la valeur MAC calculée par le destinataire doit être la même que la valeur du champ mac. Similairement, pour que l'authentification réussisse quand le champ authAttrs est présent, le hash du contenu calculé par le destinataire doit être le même que le hash inclus dans l'attribut message-digest.

Si AuthenticatedData inclus authAttrs, la valeur de l'attribut content-type doit matcher la valeur eContentType de encapContentInfo de AuthenticatedData.

Types utiles

Cette section est divisée en 2 parties. La première définis les identifiants d'algorithme, et la seconde définie d'autres type utiles.

DigestAlgorithmIdentifier

Le type DigestAlgorithmIdentifier identifie un algorithme de hashage. (ex : SHA-1, MD2, et MD5). Un algorithme de hashage mappe un chaîne d'octets (le contenu) en une autre chaîne d'octets (le hash).

DigestAlgorithmIdentifier : := AlgorithmIdentifier

SignatureAlgorithmIdentifier

Le type SignatureAlgorithmIdentifier identifie un algorithme de signature, et peut également identifier un algorithme de hashage. (ex : RSA, DSA, DSA avec SHA-1, ECDSA, et ECDSA avec SHA-256). Un algorithme de signature supporte les opérations de génération et de vérification de signature. L'opération de génération de signature utilise le hash et la clé privée du signataire pour générer une valeur signature. L'opération de vérification de signature utilise le hash et la clé publique du signataire pour déterminer si la signature est valide.

SignatureAlgorithmIdentifier : := AlgorithmIdentifier

KeyEncryptionAlgorithmIdentifier

Le type KeyEncryptionAlgorithmIdentifier identifie un algorithme de chiffrement de clé pour chiffrement une clé de chiffrement de contenu. L'opération de chiffrement mappe une chaîne d'octets (la clé) en une autre chaîne d'octets (la clé chiffrée) sous le contrôle de la clé de chiffrement. L'opération de déchiffrement est l'inverse de l'opération de chiffrement. Les détails du chiffrement et du déchiffrement dépendent de l'algorithme de gestion de clé utilisé.

KeyEncryptionAlgorithmIdentifier : := AlgorithmIdentifier

ContentEncryptionAlgorithmIdentifier

Le type ContentEncryptionAlgorithmIdentifier identifie un algorithme de chiffrement de contenu. (ex : Triple-DES et RC2). Un algorithme de chiffrement de contenu supporte les opérations de chiffrement et de déchiffrement. L'opération de chiffrement mappe une chaîne d'octets (le texte en clair) en une autre chaîne d'octets (le texte chiffré) sous le contrôle de la clé de chiffrement de contenu. L'opération de déchiffrement est l'inverse de l'opération de chiffrement.

ContentEncryptionAlgorithmIdentifier : := AlgorithmIdentifier

MessageAuthenticationCodeAlgorithm

KeyDerivationAlgorithmIdentifier

Le type KeyDerivationAlgorithmIdentifier est spécifié dans la rfc 3211. Les algorithmes de dérivation de clé convertissent un mot de passe ou une valeur de clé secrète en une clé de chiffrement de clé.

KeyDerivationAlgorithmIdentifier : := AlgorithmIdentifier

RevocationInfoChoices

Le type RevocationInfoChoices donne un jeu d'informations de statut de révocation alternatif. Il est prévu que le jeu contienne les informations suffisantes pour déterminer si les certificats et les attributs de certificat avec lequel le jeu est associé sont révoqués. Cependant, il peut y avoir plus d'informations de statut de révocation que nécessaire. Les listes de révocation de certificats X.509 sont la principale source d'informations de statut de révocation, mais tout autre format d'information de révocation peut être supporté. L'alternative OtherRevocationInfoFormat est fournis pour supporter tout autre format de révocation sans modification du CMS.

CertificateList peut contenir un CRL, une ARL, un Delta CRL, ou un attribut CRL. Toutes ces listes partagent une syntaxe commune. Le type CertificateList donne une CRL. les CRL sont spécifiés dans X.509, et sont prévus pour être utilisés sur l'Internet.

La définition de CertificateList est prise de X.509 :

```
RevocationInfoChoices ::= SET OF RevocationInfoChoice

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

OtherRevocationInfoFormat ::= SEQUENCE {
 otherRevInfoFormat OBJECT IDENTIFIER,
 otherRevInfo ANY DEFINED BY otherRevInfoFormat }
```

CertificateChoices

Le type CertificateChoices donne soit un certificat étendus PKCS#6, un ACv1, ACv2, ou tout autre format. L'alternative OtherCertificateFormat est fournis pour supporter tout autre format de certificat sans modification du CMS.

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

OtherCertificateFormat ::= SEQUENCE {
 otherCertFormat OBJECT IDENTIFIER,
 otherCert ANY DEFINED BY otherCertFormat }
```

CertificateSet

Le type CertificateSet fournis un jeu de certificats. Il est prévu que le jeu soit suffisant pour contenir des chemins de certification depuis une autorité racine vers tour les certificats de l'émetteur avec lequel le jeu est associé. Cependant, il peut y avoir plus de certificats que nécessaire, ou moins que nécessaire.

La signification précise d'un chemin de certification est hors du scope de ce document. Certaines applications peuvent imposer des limites maximales sur la longueur d'un chemin de certification ; d'autre peuvent forcer certaines relations entre les sujets et les fournisseurs de certificats dans un chemin de certification.

CertificateSet : := SET OF CertificateChoices

IssuerAndSerialNumber

Le type IssuerAndSerialNumber identifie un certificat, et donc une entité et une clé publique, par le nom distinct du fournisseur du certificat et un numéro de série de certificat spécifique au fournisseur.

La définition de Name vient de X.501-88 et de CertificateSerialNumber vient de X.509-97 :

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

CertificateSerialNumber ::= INTEGER
```

CMSVersion

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

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

UserKeyingMaterial

Le type UserKeyingMaterial donne une syntaxe pour UKM. Certains agréments de clé nécessitent UKM pour s'assurer qu'une clé différente est générée à chaque fois que les même 2 parties génèrent une clé pairée. L'émetteur fournis un UKM à utiliser avec un algorithme d'agrément de clé spécifique.

UserKeyingMaterial : := OCTET STRING

OtherKeyAttribute

Le type OtherKeyAttribute donne une syntaxe pour inclure d'autres attributs de clé qui permettent au destinataire de sélectionner la clé utilisé par l'émetteur. L'identifiant d'objet d'attribut doit être enregistré avec la syntaxe de l'attribut lui-même. L'utilisatino de cette structure devrait être évité vu qu'il peut entraver l'intéropérabilité

```
OtherKeyAttribute ::= SEQUENCE {
 keyAttrId OBJECT IDENTIFIER,
 keyAttr ANY DEFINED BY keyAttrId OPTIONAL }
```

Attributs utiles

Cette section définis les attributs qui peuvent être utilisés avec signed-data, enveloped-data, encrypted-data, ou authenticated-data. La syntaxe de l'attribut est compatible avec X.501. Certains des attributs définis dans cette section ont été définis à l'origine dans PKCS#9 ; d'autres ont été définis dans une précédente version de cette spécification. Les attributs ne sont pas listés dans un ordre particulier.

D'autres attributs sont définis ailleurs, notamment dans S/MIME v3.1 et dans ESS, qui inclus également des recommandations sur le placement de ces attributs.

Content Type

Le type d'attribut content-type spécifie le type de contenu de ContentInfo dans une donnée signée ou une donnée authentifié. L'attribut content-type doit être présent quand des attributs signés sont présents dans une donnée signée ou que des attributs authentifiés sont présents dans une donnée authentifiée. La valeur de l'attribut content-type doit matcher la valeur eContentType de encapContentInfo dans la donnée signée ou la donnée authentifiée.

L'attribut content-type doit être un attribut signé ou un attribut authentifié ; il ne doit pas être un attribut non-signé, non-authentifié, ou non-protégé.

L'identifiant d'objet suivant identifie l'attribut content-type :

```
id-contentType OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }
```

Les valeurs d'attribut content-type ont le type ASN.1 :

```
ContentType ::= OBJECT IDENTIFIER
```

Même si la syntaxe est définie comme un jeu d'AttributeValue, un attribut content-type doit avoir un simple valeur d'attribut. 0 ou plusieurs instances de AttributeValue ne sont pas permis.

Les syntaxe SignedAttributes et AuthAttributes sont chacun définis comme un jeu d'attributs. SignedAttributes dans un signerInfo ne doit pas inclure plusieurs instances de l'attribut content-type. Similairement, AuthAttributes dans un AuthenticatedData ne doit pas inclure plusieurs instances de l'attribut content-type.

Message Digest

Le type d'attribut message-digest spécifie la hash de la chaîne d'octet eContent de encapContentInfo à signer dans une donnée signée, ou à authentifier dans une donnée authentifiée. pour une donnée signée, le hash est calculé en utilisant l'algorithme de hashage du signataire. Pour une donnée authentifiée, le hash est calculé en utilisant l'algorithme de hashage de l'auteur.

Dans une donnée signée, le type d'attribut signé message-digest doit être présent quand il y a des attributs signés. dans une donnée authentifiée, le type d'attribut authentifié message-digest doit être présent quand il à des attribut authentifiés.

L'attribut message-digest doit être un attribut signé ou un attribut authentifié ; il ne doit pas être un attribut non-signé, non-authentifié, ou non-protégé.

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

```
id-messageDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }
```

Les valeurs d'attribut message-digest ont le type ASN.1 :

```
MessageDigest ::= OCTET STRING
```

Un attribut message-digest doit avoir une valeur d'attribut simple, même si la syntaxe est définie comme un jeu d'AttributValue. Il ne doit pas y avoir 0 ou plusieurs instances de AttributeValue.

La syntaxe SignedAttributes et AuthAttributes sont chacun définis comme jeu d'attributs. SignedAttributes dans un signerInfo doit inclure seulement une instance de l'attribut message-digest. Similairement, AuthAttributes dans un AuthenticatedData doit inclure seulement une instance de l'attribut message-digest.

Signing Time

Le type d'attribut signing-time spécifie la date à laquelle le signataire a effectué la signature. Le type d'attribut signing-time est prévu pour être utilisé dans une donnée signée. Il doit être un attribut signé, ou un attribut authentifié.

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

```
id-signingTime OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }
```

Les valeurs de l'attribut signing-time ont le type ASN.1 :

```
SigningTime ::= Time

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

Note : la définition de Time matche celui spécifié dans la version 1997 de X.509. Les dates entre le 1 janvier 1950 et le 31 décembre 2049 (inclus) doit être encodé en UTCTime. Toutes dates en dehors de cette plage doivent être encodés en GeneralizedTime.

Les valeurs UTCTime doivent être exprimés en Coordinated Universal Time (GMT) et doivent inclure les secondes (YYMMDDHHMMSSZ), même où le nombre de secondes est 0. Minuit doit être représenté "YYMMDD000000Z". L'information du siècle est implicit, et le siècle doit être interprété comme suit :

- où YY est supérieur ou égal à 50, l'année doit être interprétée en 19YY
- où YY est inférieur ou égal à 50, l'année doit être interprétée en 20YY.

Les valeurs GeneratizedTime doivent être exprimées en GMT et doivent inclure les secondes (YYYYMMDDHHMMSSZ), même quand le nombre de seconde est 0. Les valeurs GeneralizedTime ne doivent pas inclure des fractions de secondes.

Un attribut signing-time doit avoir une valeur d'attribut simple, même si la syntaxe est définie comme jeu d'AttributeValue. Il ne doit pas y avoir 0 ou plusieurs instances d'AttributeValue.

La syntaxe SignedAttributes et la syntaxe AuthAttributes sont chacun définis comme jeu d'attributs. SignedAttributes dans un signerInfo ne doit pas inclure plusieurs instances de l'attribut signing-time. Similairement, AuthAttributes dans un AuthenticatedData ne doit pas inclure plusieurs instances de l'attribut signing-time.

Aucun prérequis n'est imposé concernant l'exactitude de la date de signature, et l'acceptation d'une date de signature est laissée à la discretion du destinataire. Il est prévu, cependant, que certains signataires, tels que les serveurs time-stamp, vont l'approuver implicitement.

Countersignature

Le type d'attribut Countersignature spécifie une ou plusieurs signatures sur les octets de contenu de la chaîne d'octets signature dans une valeur signerInfo d'une donnée signée. C'est à dire que le hash est calculé sur les octets comprenant la valeur de la chaîne d'octets, sans inclure le tag ni les octets de longueur. Donc, le type d'attribut countersignature contre-signe (signe en série) une autre signature.

L'attribut countersignature 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 countersignature :

```
id-countersignature OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 6 }
```

Les valeurs d'attribut countersignature ont le type ASN.1 :

```
Countersignature ::= SignerInfo
```

Les valeurs countersignature ont la même signification que les valeurs SignerInfo pour les signatures ordinaires, excepté :

1. le champ signedAttributes ne doit pas contenir un attribut content-type, il n'y a pas de type de contenu pour les contre-signatures
2. Le champ signedAttributes doit contenir un attribut message-digest s'il contient d'autres attributs
3. L'entrée du processus de hashage sont les octets de contenu de l'encodé DER du champ signatureValue de la valeur SignerInfo avec lequel l'attribut est associé.

Un attribut countersignature peut avoir plusieurs valeurs d'attribut. La syntaxe est définie comme jeu d'AttributeValue, et il doit y avoir au moins une valeur présente.

La syntaxe UnsignedAttributes est définie comme un jeu d'attributs. UnsignedAttributes dans un SignerInfo peut inclure plusieurs instances de l'attribut countersignature.

Un countersignature, vu qu'il a le type SignerInfo, peut lui-même contenir un attribut countersignature. Donc, il est possible de construire arbitrairement une longue série de contre-signatures.

Modules ASN.1

Cette section contient le module ASN.1 pour le CMS, et la section suivante contient le module ASN.1 pour l'attribut certificat version 1.

Module ASN.1 CMS

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

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

- EXPORTS All
- Les types et valeurs définies dans ce module sont exportés pour l'utilisation dans
- d'autres modules ASN.1. D'autres applications peuvent les utiliser pour leur propre usage.

IMPORTS

 - Imports de la RFC 5280, Appendix A.1
  AlgorithmIdentifier, Certificate, CertificateList,
  CertificateSerialNumber, Name
   FROM PKIX1Explicit88
    { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) mod(0)
pkix1-explicit(18) }

 - Imports de la RFC 3281, Appendix B
  AttributeCertificate
   FROM PKIXAttributeCertificate
    { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) mod(0)
attribute-cert(12) }

 - Imports de l'Appendix B de ce document
```

```
  AttributeCertificateV1
   FROM AttributeCertificateVersion1
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) v1AttrCert(15) } ;

- Cryptographic Message Syntax

ContentInfo ::= SEQUENCE {
 contentType ContentType,
 content [0] EXPLICIT ANY DEFINED BY contentType }

ContentType ::= OBJECT IDENTIFIER

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

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo

EncapsulatedContentInfo ::= SEQUENCE {
 eContentType ContentType,
 eContent [0] EXPLICIT OCTET STRING OPTIONAL }

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

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

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute

UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
 attrType OBJECT IDENTIFIER,
 attrValues SET OF AttributeValue }

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING

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

```
OriginatorInfo ::= SEQUENCE {
 certs [0] IMPLICIT CertificateSet OPTIONAL,
 crls [1] IMPLICIT RevocationInfoChoices OPTIONAL }

RecipientInfos ::= SET SIZE (1..MAX) OF RecipientInfo

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

EncryptedContent ::= OCTET STRING

UnprotectedAttributes ::= SET SIZE (1..MAX) OF Attribute

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

EncryptedKey ::= OCTET STRING

KeyTransRecipientInfo ::= SEQUENCE {
 version CMSVersion, - always set to 0 or 2
 rid RecipientIdentifier,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }

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

KeyAgreeRecipientInfo ::= SEQUENCE {
 version CMSVersion, - always set to 3
 originator [0] EXPLICIT OriginatorIdentifierOrKey,
 ukm [1] EXPLICIT UserKeyingMaterial OPTIONAL,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 recipientEncryptedKeys RecipientEncryptedKeys }

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

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

RecipientEncryptedKeys ::= SEQUENCE OF RecipientEncryptedKey

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

KeyAgreeRecipientIdentifier ::= CHOICE {
 issuerAndSerialNumber IssuerAndSerialNumber,
```

```
 rKeyId [0] IMPLICIT RecipientKeyIdentifier }

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

SubjectKeyIdentifier ::= OCTET STRING

KEKRecipientInfo ::= SEQUENCE {
 version CMSVersion, - always set to 4
 kekid KEKIdentifier,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }

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

PasswordRecipientInfo ::= SEQUENCE {
 version CMSVersion, - always set to 0
 keyDerivationAlgorithm [0] KeyDerivationAlgorithmIdentifier OPTIONAL,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }

OtherRecipientInfo ::= SEQUENCE {
 oriType OBJECT IDENTIFIER,
 oriValue ANY DEFINED BY oriType }

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

Digest ::= OCTET STRING

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

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

AuthAttributes ::= SET SIZE (1..MAX) OF Attribute

UnauthAttributes ::= SET SIZE (1..MAX) OF Attribute

MessageAuthenticationCode ::= OCTET STRING
```

```
DigestAlgorithmIdentifier ::= AlgorithmIdentifier

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier

KeyDerivationAlgorithmIdentifier ::= AlgorithmIdentifier

RevocationInfoChoices ::= SET OF RevocationInfoChoice

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

OtherRevocationInfoFormat ::= SEQUENCE {
 otherRevInfoFormat OBJECT IDENTIFIER,
 otherRevInfo ANY DEFINED BY otherRevInfoFormat }

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

AttributeCertificateV2 ::= AttributeCertificate

OtherCertificateFormat ::= SEQUENCE {
 otherCertFormat OBJECT IDENTIFIER,
 otherCert ANY DEFINED BY otherCertFormat }

CertificateSet ::= SET OF CertificateChoices

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

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

UserKeyingMaterial ::= OCTET STRING

OtherKeyAttribute ::= SEQUENCE {
 keyAttrId OBJECT IDENTIFIER,
 keyAttr ANY DEFINED BY keyAttrId OPTIONAL }

- Content Type Object Identifiers

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

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

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

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

id-digestedData OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 5 }

id-encryptedData OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 6 }

id-ct-authData OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) ct(1) 2 }

- The CMS Attributes

MessageDigest ::= OCTET STRING

SigningTime ::= Time

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

Countersignature ::= SignerInfo

- Attribute Object Identifiers

id-contentType OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }

id-messageDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }

id-signingTime OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }

id-countersignature OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 6 }

- Obsolete Extended Certificate syntax from PKCS #6

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

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

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

Signature ::= BIT STRING

END - of CryptographicMessageSyntax2004
```

Module ASN.1 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 EXPLICIT TAGS ::=
BEGIN

- EXPORTS All

IMPORTS

 - Imports from RFC 5280 [PROFILE], Appendix A.1
  AlgorithmIdentifier, Attribute, CertificateSerialNumber,
  Extensions, UniqueIdentifier
   FROM PKIX1Explicit88
    { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) mod(0)
pkix1-explicit(18) }

 - Imports from RFC 5280 [PROFILE], Appendix A.2
  GeneralNames
   FROM PKIX1Implicit88
    { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) mod(0)
pkix1-implicit(19) }

 - Imports from RFC 3281 [ACPROFILE], Appendix B
  AttCertValidityPeriod, IssuerSerial
   FROM 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.509-1997, mais des noms de type différents sont utilisés pour éviter les
collisions

AttributeCertificateV1 ::= SEQUENCE {
 acInfo AttributeCertificateInfoV1,
 signatureAlgorithm AlgorithmIdentifier,
 signature BIT STRING }

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

AttCertVersionV1 ::= INTEGER { v1(0) }

END - of AttributeCertificateVersion1
```