Groupe de travail Réseau
Request for Comments : 2634
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

P. Hoffman, éditeur
Internet Mail Consortium
juin 1999

Services de sécurité améliorée pour S/MIME

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.

Table des Matières

1. Introduction

Le présent document décrit quatre extensions facultatives de service de sécurité pour S/MIME. Les services sont :
- des récépissés signés
- des étiquettes de sécurité
- des listes de diffusion de message sécurisées
- des certificats signés

Les trois premiers services fournissent une fonctionnalité qui est similaire à celle du protocole de sécurité de message [MSP4], mais qui est utile dans de nombreux autres environnements, en particulier des affaires et de la finance. Signer les certificats est utile dans tout environnement où des certificats pourraient être transmis avec des messages signés.

Les services décrits ici sont des extensions à S/MIME version 3 ([RFC2633] et [RFC2632]), certains d'entre eux peuvent aussi être ajoutés à S/MIME version 2 [RFC2311]. Les extensions décrites ici ne causeront pas d'incapacité pour un receveur de S/MIME version 3 de lire des messages d'un envoyeur de S/MIME version 2. Cependant, certaines des extensions feront que des messages créés par un envoyeur de S/MIME version 3 seront illisibles par un receveur de S/MIME version 2.

Le présent document décrit à la fois les procédures et les attributs nécessaires pour les quatre services. Noter que certains des attributs décrits dans le document sont assez utiles dans d'autres contextes et devraient être pris en considération lors de l'extension de S/MIME ou d'autres applications de CMS.

Le format des messages est décrit en ASN.1:1988 [X.208].

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

1.1 Triple enveloppe

Certaines des caractéristiques de chaque service utilisent le concept de message à "triple enveloppe". Un message à triple enveloppe est celui qui a été signé, puis chiffré, puis signé à nouveau. Les signataires des signatures interne et externe peuvent être des entités différentes ou la même entité. Noter que la spécification S/MIME ne limite pas le nombre d'encapsulations incorporées, de sorte qu'il peut y avoir plus de trois enveloppes.

1.1.1 Objet de la triple enveloppe

Tous les messages n'ont pas besoin d'avoir une triple enveloppe. La triple enveloppe est utilisée lorsque un message doit être signé, puis chiffré, et avoir ensuite des attributs signés liés au corps chiffré. Des attributs externes peuvent être ajoutés ou retirés par le générateur du message ou des agents intermédiaires, et peuvent être signés par des agents intermédiaires ou par le receveur final.

La signature intérieure est utilisée pour l'intégrité du contenu, la non répudiation avec preuve d'origine, et pour lier des attributs (comme une étiquette de sécurité) au contenu original. Ces attributs vont de l'origine au receveur, sans considération du nombre des entités intermédiaires, telles que les agents de liste de messagerie, qui traitent le message. Les attributs signés peuvent être utilisés pour le contrôle d'accès au corps interne. Les demandes de récépissés signés par le générateur sont portées aussi dans la signature interne.

Le corps chiffré assure la confidentialité, y compris la confidentialité des attributs qui sont portés dans la signature interne.

La signature externe assure l'authentification et l'intégrité des informations qui sont traitées bond par bond, où chaque bond est une entité intermédiaire telle qu'un agent de liste de messagerie. La signature externe lie les attributs (tels qu'une étiquette de sécurité) au corps chiffré. Ces attributs peuvent être utilisés pour le contrôle d'accès et les décisions d'acheminement.

1.1.2 Étapes de la triple enveloppe

Les étapes pour créer un message à triple enveloppe sont :

1. Commencer par un corps de message, appelé le "contenu d'origine".

2. Encapsuler le contenu d'origine avec les en-têtes MIME Content-type appropriés, tel que "Content-type: text/plain". Une

exception à cette règle d'encapsulation MIME est qu'un récépissé signé n'est pas mis dans les en-têtes MIME.

3. Signer le résultat de l'étape 2 (les en-têtes MIME internes et le contenu d'origine). L'identifiant d'objet SignedData encapContentInfo eContentType DOIT être id-data. Si la structure créée à l'étape 4 est multipartie/signée, alors le SignedData encapContentInfo eContent DOIT être absent. Si la structure créée à l'étape 4 est application/pkcs7-mime, alors le SignedData encapContentInfo eContent DOIT contenir le résultat de l'étape 2 ci-dessus. La structure SignedData est encapsulée par une séquence ContentInfo avec un type de contenu de id-signedData.

4. Ajouter une construction MIME appropriée au message signé de l'étape 3 comme défini dans la [RFC2633]. Le message résultant est appelé la "signature interne".
   - Si on signe en utilisant un multipartie/signé, la construction MIME ajoutée consiste en un type de contenu de multipart/signed avec paramètres, le délimiteur, le résultat de l'étape 2 ci-dessus, le délimiteur, un type de contenu de application/pkcs7-signature, des en-têtes MIME facultatifs (tels que Content-transfer-encoding et Content-disposition) et une partie de corps qui est le résultat de l'étape 3 ci-dessus.
   - Si on signe en utilisant application/pkcs7-mime, la construction MIME ajoutée consiste en un type de contenu de application/pkcs7-mime avec des paramètres, des en-têtes MIME facultatifs (tels que Content-transfer-encoding et Content-disposition) et le résultat de l'étape 3 ci-dessus.

5. Chiffrer le résultat de l'étape 4 comme un seul bloc, le transformant en un objet application/pkcs7-mime. Le type de contenu EnvelopedData encryptedContentInfo DOIT être id-data. La structure EnvelopedData est encapsulée par une séquence ContentInfo avec un type de contenu de id-envelopedData. Ceci est appelé le "corps chiffré".

6. Ajouter les en-têtes MIME appropriés : un type de contenu de application/pkcs7-mime avec des paramètres, et des en-têtes MIME facultatifs tels que Content-transfer-encoding et Content-disposition.

7. En utilisant la même logique que dans l'étape 3 ci-dessus, signer le résultat de l'étape 6 (les en-têtes MIME et le corps chiffré) comme un seul bloc

8. En utilisant la même logique que dans l'étape 4 ci-dessus, ajouter une construction MIME appropriée au message signé de l'étape 7. Le message résultant est appelé la "signature externe", et est aussi le message à triple enveloppe.

1.2 Format d'un message à triple enveloppe

Un message à triple enveloppe a de nombreuses couches d'encapsulation. La structure diffère sur la base du choix du format des portions signées du message. À cause de la façon dont MIME encapsule les données, les couches n'apparaissent pas dans l'ordre, et la notion de "couches" devient vague.

Il n'est pas nécessaire d'utiliser le format multipart/signed dans une signature interne parce qu'on sait que le receveur est capable de traiter les messages S/MIME (parce qu'il a déchiffré l'enveloppe du milieu). Un agent envoyeur peut choisir d'utiliser le format multipart/signed dans la couche externe afin qu'un agent non S/MIME puisse voir que la prochaine couche interne est chiffrée ; cependant, ceci n'est pas d'une grande valeur, car tout ce que cela montre au receveur est que le reste du message est illisible. Comme de nombreux agents envoyeurs utilisent toujours des structures multipart/signed, tous les agents receveurs DOIVENT être capables d'interpréter les structures de signature multipart/signed ou application/pkcs7-mime.

Le format d'un message à triple enveloppe qui utilise multipart/signed pour les deux signatures est :

```
[étape 8] Type de contenu : multipart/signed;
[étape 8] protocole ="application/pkcs7-signature";
[étape 8] délimiteur=délimiteur externe
[étape 8]
[étape 8] --délimiteur externe
[étape 6] Type de contenu : application/pkcs7-mime;          )
[étape 6]    smime-type=enveloped-data                   )
[étape 6]                                          )
[étape 4] Type de contenu : multipart/signed;              | )
[étape 4] protocole ="application/pkcs7-signature";     | )
[étape 4] délimiteur = délimiteur interne                  | )
[étape 4]                                       | )
[étape 4] --délimiteur interne                          | )
[étape 2] Type de contenu : text/plain                  % | )
[étape 2]                                       % | )
```

[étape 1] Contenu d'origine % |)
[étape 4] |)
[étape 4] --délimiteur interne |)
[étape 4] Type de contenu : application/pkcs7-signature |)
[étape 4] |)
[étape 3] bloc interne SignedData (eContent manque) |)
[étape 4] |)
[étape 4] --délimiteur interne-- |)
[étape 8]
[étape 8] --délimiteur externe
[étape 8] Type de contenu : application/pkcs7-signature
[étape 8]
[étape 7] bloc externe SignedData (eContent manque)
[étape 8]
[étape 8] --délimiteur externe--

% = Ces lignes sont celles sur lesquelles la signature interne est calculée.
| = Ces lignes sont celles qui sont chiffrées à l'étape 5. Ce résultat chiffré est opaque et est une partie d'un bloc EnvelopedData.
) = Ces lignes sont celles sur lesquelles la signature externe est calculée.

Le format d'un message à triple enveloppe qui utilise application/pkcs7-mime pour les deux signatures est :

[étape 8] Type de contenu : application/pkcs7-mime;
[étape 8] smime-type=signed-data
[étape 8]
[étape 7] bloc externe SignedData (eContent est présent) O
[étape 6] Type de contenu : application/pkcs7-mime;) O
[étape 6] smime-type=enveloped-data;) O
[étape 6]) O
[étape 4] Type de contenu : application/pkcs7-mime; |) O
[étape 4] smime-type=signed-data |) O
[étape 4] |) O
[étape 3] bloc interne SignedData (eContent est présent) I |) O
[étape 2] Type de contenu : text/plain I |) O
[étape 2] I |) O
[étape 1] Contenu d'origine I |) O

I = Ces lignes sont le bloc interne SignedData, qui est opaque et contient le résultat codé en ASN.1 de l'étape 2 ainsi que des informations de contrôle.
| = Ces lignes sont celles qui sont chiffrées à l'étape 5. Ce résultat chiffré est opaque et fait partie d'un bloc EnvelopedData.
) = Ces lignes sont celles sur lesquelles la signature externe est calculée.
O = Ces lignes sont le bloc externe SignedData, qui est opaque et contient le résultat codé en ASN.1 de l'étape 6 ainsi que des informations de contrôle.

1.3 Services de sécurité et triple enveloppe

Les trois premiers services de sécurité décrits dans ce document sont utilisés de différentes façons avec les messages à triple enveloppe. Ce paragraphe décrit brièvement les relations de chaque service avec le triple enveloppement ; les autres paragraphes de la section rentrent dans les détails.

1.3.1 Récépissés signés et triple enveloppe

Un récépissé signé peut être demandé dans tout objet SignedData. Cependant, si un récépissé signé est demandé pour un message à triple enveloppe, la demande de récépissé DOIT être dans la signature interne, et non dans la signature externe. Un agent de liste de diffusion sûr peut changer la politique de récépissé dans la signature externe d'un message à triple enveloppe lorsque ce message est traité par la liste de diffusion.

Note : les récépissés signés et les demandes de récépissés décrits dans le présent mémoire diffèrent de ceux décrits dans les travaux du groupe de travail Notification de réception de l'IETF. Le résultat de ce groupe de travail, lorsque il sera achevé, n'est pas supposé bien interagir avec les messages à triple enveloppe décrits dans le présent document.

1.3.2 Étiquettes de sécurité et triple enveloppe

Une étiquette de sécurité peut être incluse dans les attributs signés de tout objet SignedData. Un attribut d'étiquette de sécurité peut être inclus dans la signature interne, dans la signature externe, ou dans les deux.

L'étiquette de sécurité interne est utilisée pour les décisions de contrôle d'accès en rapport avec le contenu d'origine du texte source. La signature interne assure l'authentification et protège cryptographiquement l'intégrité de l'étiquette de sécurité du signataire d'origine qui est dans le corps interne. Cette stratégie facilite la transmission des messages parce que l'étiquette de sécurité du signataire d'origine est incluse dans le bloc SignedData qui peut être transmis à un tiers qui peut vérifier la signature interne couvrant l'étiquette de sécurité interne. Le service de sécurité de confidentialité peut être appliqué à l'étiquette de sécurité interne en chiffrant le bloc SignedData interne entier au sein d'un bloc EnvelopedData.

Une étiquette de sécurité peut aussi être incluse dans les attributs signés du bloc SignedData block externe qui va inclure les sensibilités du message chiffré. L'étiquette de sécurité externe est utilisée pour les décisions de contrôle d'accès et d'acheminement qui se rapportent au message chiffré. Noter qu'un attribut d'étiquette de sécurité ne peut être utilisé que dans un bloc signedAttributes. Un attribut eSSSecurityLabel NE DOIT PAS être utilisé dans des attributs EnvelopedData ou non signés.

1.3.3 Listes de messagerie sécurisées et triple enveloppe

Le traitement du message de liste de diffusion sécurisée dépend de la structure des couches S/MIME présentes dans le message envoyé à l'agent de liste de diffusion. L'agent de liste de diffusion ne change jamais les données qui ont été hachées pour former la signature interne, si une telle signature est présente. Si une signature externe est présente, l'agent va alors modifier les données qui ont été hachées pour former cette signature externe. Dans tous les cas, l'agent ajoute ou met à jour un attribut mlExpansionHistory pour documenter le traitement de l'agent, et ajoute ou remplace finalement la signature externe sur le message à distribuer.

1.3.4 Placement des attributs

Certains attributs devraient être placés dans le message SignedData interne ou externe ; certains attributs peuvent être dans l'un et l'autre. De plus, certains attributs doivent être signés tandis que la signature est facultative pour d'autres, et que certains attributs ne doivent pas être signés. ESS définit plusieurs types d'attributs. ContentHints et ContentIdentifier PEUVENT apparaître dans toute liste d'attributs. contentReference, equivalentLabel, eSSSecurityLabel et mlExpansionHistory DOIVENT être portés dans un type SignedAttributes ou AuthAttributes, et NE DOIVENT PAS être portés dans des types d'attributs UnauthAttributes ou UnprotectedAttributes. msgSigDigest, receiptRequest et signingCertificate DOIVENT être portés dans un SignedAttributes, et NE DOIVENT PAS être portés dans un type AuthAttributes, UnsignedAttributes, UnauthAttributes ou UnprotectedAttributes.

Le tableau suivant résume la recommandation de ce profil. Dans la colonne OID, [ESS] indique que l'attribut est défini dans le présent document.

Attribut	OID	Interne ou externe	Signé
contentHints	id-aa-contentHint [ESS]	l'un ou l'autre	PEUT
contentIdentifier	id-aa-contentIdentifier [ESS]	l'un ou l'autre	PEUT
contentReference	id-aa-contentReference [ESS]	l'un ou l'autre	DOIT
contentType	id-contentType [RFC2630]	l'un ou l'autre	DOIT
counterSignature	id-countersignature [RFC2630]	l'un ou l'autre	NE DOIT PAS
equivalentLabel	id-aa-equivalentLabels [ESS]	l'un ou l'autre	DOIT
eSSSecurityLabel	id-aa-securityLabel [ESS]	l'un ou l'autre	DOIT
messageDigest	id-messageDigest [RFC2630]	l'un ou l'autre	DOIT
msgSigDigest	id-aa-msgSigDigest [ESS]	seulement interne	DOIT
mlExpansionHistory	id-aa-mlExpandHistory [ESS]	seulement externe	DOIT
receiptRequest	id-aa-receiptRequest [ESS]	seulement interne	DOIT
signingCertificate	id-aa-signingCertificate [ESS]	l'un ou l'autre	DOIT
signingTime	id-signingTime [RFC2630]	l'un ou l'autre	DOIT
smimeCapabilities	sMIMECapabilities [RFC2633]	l'un ou l'autre	DOIT
sMIMEEncryption-KeyPreference	id-aa-encrypKeyPref [RFC2633]	l'un ou l'autre	DOIT

La CMS définit signedAttrs comme un ENSEMBLE d'attributs et définit unsignedAttrs comme ENSEMBLE d'attributs. ESS définit les types d'attributs contentHints, contentIdentifier, eSSecurityLabel, msgSigDigest, mlExpansionHistory, receiptRequest, contentReference, equivalentLabels et signingCertificate. Un signerInfo NE DOIT PAS inclure plusieurs instances d'un des types d'attributs définis dans ESS. Les paragraphes suivants de ESS spécifient d'autres restrictions qui s'appliquent aux types d'attributs receiptRequest, mlExpansionHistory et eSSecurityLabel.

La CMS définit la syntaxe des attributs signés et non signés comme "attrValues SET OF AttributeValue". Pour tous les types d'attributs définis dans ESS, si le type d'attribut est présent dans un signerInfo, il DOIT alors inclure seulement une instance de AttributeValue. En d'autres termes, il NE DOIT PAS y avoir zéro ou plusieurs instances de AttributeValue présentes dans le "attrValues SET OF AttributeValue".

Si un attribut counterSignature est présent, il DOIT alors être inclus dans les attributs non signés. Il NE DOIT PAS être inclus dans les attributs signés. Les seuls attributs qui sont permis dans un attribut counterSignature sont counterSignature, messageDigest, signingTime, et signingCertificate.

Noter que les signatures internes et externes sont usuellement celles des différents envoyeurs. À cause de cela, le même attribut dans les deux signatures pourrait conduire à des conséquences très différentes.

ContentIdentifier est un attribut (CHAINE D'OCTETS) utilisé pour porter un identifiant unique alloué au message.

1.4 Attributs exigés et facultatifs

Certaines passerelles de sécurité signent des messages qui passent à travers elles. Si le message est d'un type autre que signedData, la passerelles a un seul moyen de signer le message : en l'enveloppant dans un bloc signedData et des en-têtes MIME. Si le message à signer par la passerelle est déjà un message signedData, la passerelle peut signer le message en insérant un signerInfo dans le bloc signedData.

Le principal avantage de l'ajout par une passerelle d'un signerInfo au lieu d'envelopper le message dans une nouvelle signature est que le message ne grossit pas autant que si la passerelle enveloppe le message. Le principal inconvénient est que la passerelle doit vérifier la présence de certains attributs dans l'autre signerInfos, et soit omettre, soit copier, ces attributs.

Si une passerelle ou un autre processeur ajoute un signerInfo à un bloc signedData existant, elle DOIT copier les attributs mlExpansionHistory et eSSSecurityLabel provenant des autres signerInfos. Cela aide à s'assurer que le receveur va traiter ces attributs dans un signerInfo qu'il peut vérifier.

Noter que quelqu'un peut à l'avenir définir un attribut qui doive être présent dans chaque signerInfo d'un bloc signedData afin que la signature soit traitée. Si cela arrivait, une passerelle qui insère un signerInfos et ne copie pas cet attribut va causer l'échec de tout message qui a cet attribut lors du traitement par le receveur. Pour cette raison, il est plus sûr d'envelopper les messages avec de nouvelles signatures plutôt que d'insérer des signerInfos.

1.5 Identifiants d'objet

Les identifiants d'objets pour beaucoup des objets décrits dans le présent mémoire se trouvent dans les [RFC2630], [RFC2633], et [RFC2632]. D'autres identifiants d'objets utilisés dans S/MIME peuvent être trouvés dans le registre tenu à <http://www.imc.org/ietf-smime/oids.html>. Lorsque le présent mémoire avancera sur la voie de la normalisation de l'IETF, il est prévu que l'IANA tienne ce registre.

2. Récépissés signés

Retourner un récépissé signé donne à l'origine une preuve de la livraison d'un message, et lui permet de démontrer à un tiers que le receveur a été capable de vérifier la signature du message original. Ce récépissé est lié au message original par la signature ; par conséquent, ce service ne peut être demandé que si un message est signé. L'envoyeur du récépissé peut aussi facultativement chiffrer un récépissé pour assurer la confidentialité entre l'envoyeur du récépissé et le receveur du récépissé.

2.1 Concepts de récépissés signés

Le générateur d'un message peut demander un récépissé signé aux receveurs d'un message. La demande est indiquée par l'ajout d'un attribut receiptRequest au champ signedAttributes de l'objet SignerInfo pour lequel le récépissé est demandé. Le logiciel d'agent d'utilisateur receveur DEVRAIT créer automatiquement un récépissé signé lorsque on lui demande de le faire, et retourner le récépissé conformément aux options d'expansion de la liste de diffusion, aux politiques de sécurité locales, et aux options de configuration.

Parce que les récépissés impliquent l'interaction de deux parties, la terminologie peut parfois être troublante. Dans cette section, l'"envoyeur" est l'agent qui envoie le message original qui incluait une demande de récépissé. Le "receveur" est la partie qui a reçu ce message et a généré le récépissé.

Les étapes d'une transaction normale sont :
1. l'envoyeur crée un message signé qui inclut un attribut de demande de récépissé (paragraphe 2.2).
2. l'envoyeur transmet le message résultant aux receveurs.
3. les receveurs du message déterminent si il y a une signature valide et une demande de récépissé dans le message (paragraphe 2.3).
4. le receveur crée un récépissé signé (paragraphe 2.4).
5. le receveur transmet le message de récépissé signé résultant à l'envoyeur (paragraphe 2.5).
6. l'envoyeur reçoit le message et valide qu'il contient un récépissé signé pour le message original (paragraphe 2.6). Cette validation s'appuie sur le fait que l'envoyeur a conservé soit une copie du message original, soit des informations extraites du message original.

La syntaxe ASN.1 pour la demande de récépissé est au paragraphe 2.7 ; la syntaxe ASN.1 du récépissé est au paragraphe 2.8.

Noter qu'un agent envoyeur DEVRAIT se souvenir de quand il a envoyé un récépissé afin de pouvoir éviter de renvoyer un récépissé chaque fois qu'il traite le message.

Une demande de récépissé peut indiquer que les récépissés sont à envoyer en de nombreux endroits, pas seulement à l'envoyeur (en fait, la demande de récépissé peut indiquer que les récépissés ne devraient même pas aller chez l'envoyeur). Afin de vérifier un récépissé, le receveur du récépissé doit être le générateur ou un receveur du message original. Donc, l'envoyeur NE DEVRAIT PAS demander que les récépissés soient envoyés à quelqu'un qui n'a pas une copie exacte du message.

2.2 Création de la demande de récépissé

Les messages S/MIME multi couches peuvent contenir plusieurs couches de SignedData. Cependant, les récépissés ne peuvent être demandés que pour la couche SignedData la plus interne dans un message S/MIME multi couches, comme un message à triple enveloppe. Un seul attribut receiptRequest peut être inclus dans le signedAttributes d'un SignerInfo.

Un attribut ReceiptRequest NE DOIT PAS être inclus dans les attributs d'un SignerInfo dans un objet SignedData qui encapsule un contenu Receipt. En d'autres termes, l'agent receveur NE DOIT PAS demander un récépissé signé pour un récépissé signé.

Un envoyeur demande des récépissés en plaçant un attribut receiptRequest dans les attributs signés d'un signerInfo comme suit :
1. une structure de données receiptRequest est créée.
2. un identifiant de contenu signé est créé pour le message et est alloué au champ signedContentIdentifier. Ce champ est utilisé pour associer le récépissé signé avec le message qui demande le récépissé signé.
3. les entités auxquelles il est demandé de retourner un récépissé signé sont notées dans le champ receiptsFrom.
4. le générateur du message DOIT remplir le champ receiptsTo avec un GeneralNames pour chaque entité à qui le receveur devrait envoyer le récépissé signé. Si le générateur du message veut que le receveur lui envoie le récépissé signé, il DOIT alors inclure un GeneralNames pour lui-même dans le champ receiptsTo. GeneralNames est une SEQUENCE DE GeneralName. receiptsTo est une SEQUENCE DE GeneralNames dans laquelle chaque GeneralNames représente une entité. Il peut y avoir plusieurs instances de GeneralName dans chaque GeneralNames. Au minimum, le générateur du message DOIT remplir le GeneralNames de chaque entité avec l'adresse à laquelle le récépissé signé devrait être envoyé. Facultativement, le générateur du message PEUT aussi remplir le GeneralNames de chaque entité avec d'autres instance de GeneralName (comme un directoryName).
5. L'attribut receiptRequest complété est placé dans le champ signedAttributes de l'objet SignerInfo.

2.2.1 Demandes de récépissés multiples

Il peut y avoir plusieurs SignerInfos au sein d'un objet SignedData, et chaque SignerInfo peut inclure des signedAttributes. Donc, un seul objet SignedData peut inclure plusieurs SignerInfo, chaque SignerInfo ayant un attribut receiptRequest. Par exemple, un générateur peut envoyer un message signé avec deux SignerInfos, l'un contenant une signature DSS, l'autre contenant une signature RSA.

Chaque receveur DEVRAIT ne retourner qu'un seul récépissé signé.

Tous les SignerInfo n'ont pas besoin d'inclure des demandes de récépissé, mais dans tous les SignerInfo qui contiennent des demandes de récépissé, les demandes de récépissé DOIVENT être identiques.

2.2.2 Informations nécessaires pour valider les récépissés signés

L'agent envoyeur DOIT conserver un ou les deux éléments suivants pour assurer la validation des récépissés signés retournés par les receveurs :

- l'objet signedData original qui demande le récépissé signé
- la valeur de résumé de signature de message utilisée pour générer la valeur originale de la signature des signerInfo des signedData et la valeur du résumé du contenu du récépissé contenant les valeurs incluses dans l'objet signedData original. Si des récépissés signés sont demandés de plusieurs receveurs, la conservation de ces valeurs de résumé est une amélioration des performances parce que l'agent envoyeur peut réutiliser les valeurs sauvegardées lors de la vérification de chaque récépissé signé retourné.

2.3 Traitement des demandes de récépissé

Une receiptRequest n'est associée qu'à l'objet SignerInfo auquel l'attribut demande de récépissé est directement rattaché. Les logiciels de réception DEVRAIENT examiner le champ signedAttributes de chaque SignerInfo pour lequel il vérifie une signature dans l'objet signedData le plus interne pour déterminer si un récépissé est demandé. Il peut en résulter que l'agent receveur traite plusieurs attributs receiptRequest inclus dans un seul objet SignedData, comme des demandes faites par les différentes personnes qui ont signé l'objet en parallèle.

Avant de traiter un attribut signé de demande de récépissé, l'agent receveur DOIT vérifier la signature du SignerInfo qui couvre l'attribut receiptRequest. Un receveur NE DOIT PAS traiter un attribut receiptRequest qui n'a pas été vérifié. Parce que tous les attributs receiptRequest dans un objet SignedData doivent être identiques, l'application receveuse traite complètement (comme décrit dans les paragraphes suivants) le premier attribut receiptRequest qu'elle rencontre dans un SignerInfo qu'elle vérifie, et elle s'assure ensuite que tous les autres attributs receiptRequest dans les signerInfo qu'elle vérifie sont identiques au premier rencontré. Si il y a des attributs ReceiptRequest vérifiés qui ne sont pas les mêmes, le logiciel de traitement NE DOIT PAS retourner de récépissé signé. Un récépissé signé DEVRAIT être retourné si un signerInfo contenant un attribut receiptRequest peut être validé, même si les autres signerInfo contenant le même attribut receiptRequest ne peuvent pas être validés parce qu'ils sont signés en utilisant un algorithme non pris en charge par l'agent.

Si un attribut receiptRequest est absent dans les attributs signés, un récépissé signé n'a alors été demandé à aucun des receveurs du message et NE DOIT PAS être créé. Si un attribut receiptRequest est présent dans les attributs signés, alors un récépissé signé a été demandé à certains ou à tous les receveurs du message. Noter que dans certains cas, un agent receveur peut recevoir deux messages presque identiques, l'un avec une demande de récépissé et l'autre sans. Dans ce cas, l'agent receveur DEVRAIT envoyer un récépissé signé pour le message qui demande un récépissé signé.

Si un attribut receiptRequest est présent dans les attributs signés, le processus suivant DEVRAIT être utilisé pour déterminer si il a été demandé à un receveur de message de retourner un récépissé signé :

1. Si un attribut mlExpansionHistory est présent dans le bloc le plus externe de signedData, effectuer une des deux étapes suivantes, fondées sur l'absence ou la présence de mlReceiptPolicy :
   - 1.1. Si une valeur de mlReceiptPolicy est absente du dernier élément MLData, une politique de récépissé de liste de diffusion n'a pas été spécifiée et le logiciel traitant DEVRAIT examiner la valeur de l'attribut receiptRequest pour déterminer si un récépissé devrait être créé et retourné.
   - 1.2. Si une valeur de mlReceiptPolicy est présente dans le dernier élément MLData, effectuer une des deux étapes suivantes, sur la base de la valeur de mlReceiptPolicy :
     - 1.2.1. Si la valeur de mlReceiptPolicy est aucune, alors la politique de récépissé de la liste de diffusion subroge la demande de récépissé signé du générateur et un récépissé signé NE DOIT PAS être créé.
     - 1.2.2. Si la valeur de mlReceiptPolicy est insteadOf ou inAdditionTo, le logiciel traitant DEVRAIT examiner la valeur de receiptsFrom de l'attribut receiptRequest pour déterminer si un récépissé devrait être créé et retourné. Si un récépissé est créé, les champs insteadOf et inAdditionTo identifient les entités auxquelles DEVRAIT être envoyé le récépissé au lieu de ou en plus du générateur.
2. Si la valeur receiptsFrom de l'attribut receiptRequest est allOrFirstTier, effectuer une des deux étapes suivantes sur la base de la valeur de allOrFirstTier :
   - 2.1. Si la valeur de allOrFirstTier est allReceipts, un récépissé signé DEVRAIT alors être créé.
   - 2.2. Si la valeur de allOrFirstTier est firstTierRecipients, effectuer les deux étapes suivantes sur la base de la présence d'un attribut mlExpansionHistory dans un bloc signedData externe :
     - 2.2.1. Si un attribut mlExpansionHistory est présent, ce receveur n'est alors pas un receveur de premier rang et un récépissé signé NE DOIT PAS être créé.
     - 2.2.2. Si un attribut mlExpansionHistory n'est pas présent, un récépissé signé DEVRAIT alors être créé.

3. Si la valeur de receiptsFrom de l'attribut receiptRequest est une receiptList :
   3.1. Si receiptList contient un des GeneralNames du receveur, un récépissé signé DEVRAIT alors être créé.
   3.2. Si receiptList ne contient pas un des GeneralNames du receveur, un récépissé signé NE DOIT alors PAS être créé.

Un diagramme des étapes ci-dessus à exécuter pour chaque signerInfo pour lequel l'agent receveur vérifie la signature serait :

0. L'attribut Demande de récépissé est il présent ?
   OUI -> 1.
   NON -> STOP
1. mlExpansionHistory est-il dans le signedData externe ?
   OUI -> 1.1.
   NON -> 2.
   1.1. mlReceiptPolicy est-il absent ?
   OUI -> 2.
   NON -> 1.2.
   1.2. Le prendre sur la base de la valeur de mlReceiptPolicy.
      aucune -> 1.2.1.
      insteadOf ou inAdditionTo -> 1.2.2.
   1.2.1. STOP.
   1.2.2. Examiner receiptsFrom pour déterminer si un récépissé devrait être créé, le créer si nécessaire, l'envoyer aux receveurs désignés par mlReceiptPolicy, puis -> STOP.
2. La valeur de receiptsFrom est-elle allOrFirstTier ?
   OUI -> Le prendre sur la base de la valeur de allOrFirstTier.
      allReceipts -> 2.1.
      firstTierRecipients -> 2.2.
   NON -> 3.
   2.1. Créer alors un récépissé, puis -> STOP.
   2.2. mlExpansionHistory est-il dans le bloc signedData externe ?
      OUI -> 2.2.1.
      NON -> 2.2.2.
   2.2.1. STOP.
   2.2.2. Créer un récépissé, puis -> STOP.
3. La valeur de receiptsFrom de receiptRequest est elle receiptList ?
   OUI -> 3.1.
   NON -> STOP.
   3.1. receiptList contient elle le receveur ?
   OUI -> Créer un récépissé, puis -> STOP.
   NON -> 3.2.
   3.2. STOP.

2.4 Création du récépissé signé

Un récépissé signé est un objet signedData qui encapsule un contenu Receipt (aussi appelé un "signedData/Receipt"). Les récépissés signés sont créés comme suit :

1. La signature des signerInfo des signedData originales qui incluent l'attribut signé receiptRequest DOIT être vérifiée avec succès avant de créer le signedData/Receipt.
   1.1. Le contenu de l'objet signedData original est résumé comme décrit dans la [RFC2630]. La valeur de résumé résultante est alors comparée avec la valeur de l'attribut messageDigest inclus dans le signedAttributes des signedData signerInfo originales. Si ces valeurs de résumé sont différentes, le processus de vérification de la signature échoue alors et le signedData/Receipt NE DOIT PAS être créé.
   1.2. Les signedAttributes (incluant messageDigest, receiptRequest, et éventuellement, d'autres attributs signés) codés en DER ASN.1 dans les signedData signerInfo originales sont résumés comme décrit dans la [RFC2630]. La valeur de résumé résultante, appelée msgSigDigest, est alors utilisée pour vérifier la signature des signedData signerInfo originales. Si la vérification de signature échoue, le signedData/Receipt NE DOIT alors PAS être créé.
2. Une structure Receipt est créée.
   2.1. La valeur du champ Version de récépissé est réglée à 1.
   2.2. L'identifiant d'objet provenant de l'attribut contentType inclus dans le signedData signerInfo original qui inclut l'attribut receiptRequest est copié dans le contentType du récépissé.
   2.3. Le signedData signerInfo receiptRequest signedContentIdentifier original est copié dans le signedContentIdentifier du récépissé.
   2.4. La valeur de signature provenant du signedData signerInfo original qui inclut l'attribut receiptRequest est copiée

dans la originatorSignatureValue du récépissé.

3. La structure du récépissé est codée en DER ASN.1 pour produire un flux de données, D1.
4. D1 est résumé. La valeur de résumé résultante est incluse comme attribut messageDigest dans le signedAttributes des signerInfo qui vont finalement contenir la valeur de signature signedData/Receipt.
5. La valeur du résumé (msgSigDigest) calculée à l'étape 1 pour vérifier la signature des signedData signerInfo originales est incluse comme attribut msgSigDigest dans le signedAttributes des signerInfo qui vont finalement contenir la valeur de signature de signedData/Receipt.
6. Un attribut contentType incluant l'identifiant d'objet id-ct-receipt DOIT être créé et ajouté aux attributs signés des signerInfo qui vont finalement contenir la valeur de signature de signedData/Receipt.
7. Un attribut signingTime indiquant l'heure de la signature de signedData/Receipt DEVRAIT être créé et ajouté aux attributs signés des signerInfo qui vont finalement contenir la valeur de signature de signedData/Receipt. D'autres attributs (sauf receiptRequest) peuvent être ajoutés aux signedAttributes des signerInfo.
8. Les signedAttributes (messageDigest, msgSigDigest, contentType et, éventuellement, d'autres) des signerInfo sont codés en DER ASN.1 et résumés comme décrit dans la [RFC2630]. La valeur de résumé résultante est utilisée pour calculer la valeur de signature qui est alors incluse dans le signedData/Receipt signerInfo.
9. Le contenu du récépissé codé en DER ASN.1 DOIT être directement codé au sein de la CHAINE D'OCTETS eContent signedData encapContentInfo définie dans la [RFC2630]. L'identifiant d'objet id-ct-receipt DOIT être inclus dans le eContentType signedData encapContentInfo. Il en résulte un seul objet codé en ASN.1 composé d'un signedData incluant le contenu du récépissé. Le type de contenu Data NE DOIT PAS être utilisé. Le contenu du récépissé NE DOIT PAS être encapsulé dans un en-tête MIME ni dans aucun autre en-tête avant d'être codé au titre de l'objet signedData.
10. Le signedData/Receipt est alors mis dans une enveloppe MIME application/pkcs7-mime avec le paramètre smime-type réglé à "signed-receipt". Cela va permettre l'identification des récépissés signés sans avoir à casser le corps ASN.1. Le paramètre smime-type va quand même être réglé normalement dans toute couche enveloppant ce message.
11. Si le signedData/Receipt doit être chiffré au sein d'un objet envelopedData, un objet signedData externe DOIT alors être créé qui encapsule l'objet envelopedData, et un attribut contentHints avec un contentType réglé à l'identifiant d'objet id-ct-receipt DOIT être inclus dans le signedData SignerInfo signedAttributes externe. Lorsque un agent receveur traite l'objet signedData externe, la présence de l'OID id-ct-receipt dans le type de contenu contentHints indique qu'un signedData/Receipt est chiffré au sein de l'objet envelopedData encapsulé par le signedData externe.

Tous les agents envoyeurs qui prennent en charge la génération de récépissés signés ESS DOIVENT fournir la capacité d'envoyer des récépissés signés chiffrés (c'est-à-dire, un signedData/Receipt encapsulé au sein d'un envelopedData). L'agent envoyeur PEUT envoyer un récépissé signé en réponse à une signedData encapsulée dans une envelopedData demandant un récépissé signé. C'est une affaire de politique locale de décider si le récépissé signé devrait ou non être chiffré. Le récépissé signé ESS inclut la valeur de résumé de message calculée pour l'objet signedData original qui demandait le récépissé signé. Si l'objet signedData original était envoyé chiffré au sein d'un objet envelopedData et si le récépissé signé ESS est envoyé non chiffré, la valeur de résumé de message calculée pour l'objet signedData chiffré original est envoyé non chiffrée. Celui qui répond devrait considérer cela pour décider de chiffrer ou non le récépissé signé ESS.

2.4.1 Attributs MLExpansionHistory et récépissés

Un attribut MLExpansionHistory NE DOIT PAS être inclus dans les attributs d'un SignerInfo dans un objet SignedData qui encapsule un contenu Receipt. Ceci est vrai parce que lorsque un SignedData/Receipt est envoyé à un agent de liste de diffusion (MLA, *Mailing List Agent*) pour distribution, le MLA doit alors toujours encapsuler le SignedData/Receipt reçu dans un SignedData externe dans lequel le MLA va inclure l'attribut MLExpansionHistory. Le MLA ne peut pas changer le signedAttributes de l'objet SignedData/Receipt reçu, de sorte qu'il ne peut pas ajouter le MLExpansionHistory au SignedData/Receipt.

2.5 Détermination des receveurs du récépissé signé

Si un récépissé signé a été créé par le procès décrit dans les paragraphes précédents, le logiciel DOIT alors utiliser le processus suivant pour déterminer à qui le récépissé signé devrait être envoyé.

1. Le champ receiptsTo doit être présent dans l'attribut receiptRequest. Le logiciel initie la séquence des receveurs avec la ou les valeurs de receiptsTo.

2. Si l'attribut MlExpansionHistory est présent dans le bloc SignedData externe, et si le dernier MLData contient une valeur de MLReceiptPolicy ou de insteadOf, le logiciel remplace alors la séquence de receveurs par les valeurs de insteadOf.

3. Si l'attribut MlExpansionHistory est présent dans le bloc SignedData externe et si le dernier MLData contient une valeur MLReceiptPolicy de inAdditionTo, le logiciel ajoute alors la ou les valeurs de inAdditionTo à la séquence des receveurs.

2.6 Validation du récépissé signé

Un récépissé signé est communiqué comme un seul objet codé en ASN.1 composé d'un objet signedData incluant directement un contenu Receipt. Il est identifié par la présence de l'identifiant d'objet id-ct-receipt dans la valeur encapContentInfo eContentType de l'objet signedData incluant le contenu Receipt.

Bien que les receveurs ne soient pas supposés envoyer plus d'un récépissé signé, les agents receveurs DEVRAIENT être capables d'accepter plusieurs récépissés signés d'un receveur.

Un signedData/Receipt est validé comme suit :

1. Décoder l'ASN.1 de l'objet signedData incluant le contenu Receipt.

2. Extraire le contentType, signedContentIdentifier, et originatorSignatureValue de la structure Receipt décodée pour identifier le signedData signerInfo original qui demandait le signedData/Receipt.

3. Acquérir la valeur de résumé de la signature du message calculée par l'envoyeur pour générer la valeur de signature incluse dans le signedData signerInfo original qui demandait le signedData/Receipt.
   3.1. Si la valeur du résumé des signatures de message calculée par l'envoyeur a été sauvegardée en local par l'envoyeur, elle doit être localisée et restituée.
   3.2. Si elle n'a pas été sauvegardée, elle doit alors être recalculée sur la base du contenu signedData et signedAttributes original comme décrit dans la [RFC2630].

4. La valeur du résumé de signature de message calculée par l'envoyeur est alors comparée avec la valeur de msgSigDigest signedAttribute incluse dans le signedData/Receipt signerInfo. Si ces valeurs de résumé sont identiques, cela prouve alors que la valeur de résumé de la signature de message calculée par le receveur sur la base de l'objet original signedData reçu est la même que celle calculée par l'envoyeur. Cela prouve que le receveur a reçu exactement le même contenu original de signedData et de signedAttributes qu'envoyé par l'envoyeur parce que c'est la seule façon dont le receveur pourrait avoir calculé la même valeur de résumé de signature de message que celle calculée par l'envoyeur. Si les valeurs de résumés sont différentes, le processus de vérification de signature de signedData/Receipt a échoué.

5. Acquérir la valeur de résumé calculée par l'envoyeur pour le contenu Receipt construit par l'envoyeur (incluant le contentType, signedContentIdentifier, et la valeur de signature qui ont été inclus dans le signedData signerInfo original qui demandait le signedData/Receipt).
   5.1. Si la valeur du résumé de contenu Receipt calculée par l'envoyeur a été sauvegardée en local par l'envoyeur, elle doit être localisée et restituée.
   5.2. Si elle n'a pas été sauvegardée, elle doit alors être recalculée. Comme décrit au paragraphe précédent, étape 2, créer une structure Receipt incluant le contentType, signedContentIdentifier et la valeur de signature qui étaient inclus dans le signedData signerInfo original qui demandait le récépissé signé. La structure Receipt est alors codée en DER ASN.1 pour produire un flux de données qui est ensuite résumé pour produire la valeur de résumé du contenu de Receipt.

6. La valeur du résumé de contenu de Receipt calculée par l'envoyeur est alors comparée avec la valeur du messageDigest signedAttribute inclus dans le signedData/Receipt signerInfo. Si ces valeurs de résumé sont identiques, cela prouve alors que les valeurs incluses dans le contenu du Receipt par le receveur sont identiques à celles qui ont été incluses dans le signedData signerInfo original qui demandait le signedData/Receipt. Cela prouve que le receveur a reçu le signedData original signé par l'envoyeur, parce que c'est le seul moyen par lequel le receveur pourrait avoir obtenu la valeur de signature du signedData signerInfo original pour l'inclure dans le contenu de Receipt. Si les valeurs de résumé sont différentes, le procès de vérification de signature du signedData/Receipt a échoué.

7. Le signedAttributes codé en DER ASN.1 du signedData/Receipt signerInfo est résumé comme décrit dans la [RFC2630].

8. La valeur de résumé résultante est alors utilisée pour vérifier la valeur de la signature incluse dans le signedData/Receipt signerInfo. Si la vérification de la signature réussit, cela prouve alors l'intégrité de signedData/receipt signerInfo signedAttributes et cela authentifie l'identité du signataire du signedData/Receipt signerInfo. Noter que le signedAttributes inclut la valeur du résumé du contenu du Receipt calculé par le receveur (attribut messageDigest) et la valeur du résumé de la signature de message calculée par le receveur (attribut msgSigDigest). Donc, la comparaison susmentionnée des valeurs de résumé générée par le receveur et générée par l'envoyeur combinées avec la vérification réussie de la signature de signedData/Receipt prouve que le receveur a reçu le contenu exact du signedData et du signedAttributes original (prouvé par l'attribut msgSigDigest) qui a été signé par l'envoyeur de l'objet signedData original (prouvé par l'attribut messageDigest). Si la vérification de signature échoue, le procès de vérification de signature de signedData/Receipt échoue alors.

Le procès de vérification de signature pour chaque algorithme de signature qui est utilisé en conjonction avec le protocole CMS est spécifique de l'algorithme. Ces processus sont décrits dans les documents spécifiques des algorithmes.

2.7 Syntaxe de la demande de récépissé

Une valeur d'attribut receiptRequest a le type ASN.1 ReceiptRequest. On n'utilise l'attribut receiptRequest qu'au sein des attributs signés associés à un message signé.

```
ReceiptRequest ::= SEQUENCE {
  signedContentIdentifier ContentIdentifier,
  receiptsFrom ReceiptsFrom,
  receiptsTo SEQUENCE SIZE (1..ub-receiptsTo)) OF GeneralNames }

ub-receiptsTo INTEGER ::= 16

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

ContentIdentifier ::= OCTET STRING

id-aa-contentIdentifier OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) id-aa(2) 7}
```

Un signedContentIdentifier DOIT être créé par le générateur du message lors de la création d'une demande de récépissé. Pour assurer l'unicité mondiale, le signedContentIdentifier minimal DEVRAIT contenir un enchaînement d'informations d'identification spécifiques de l'envoyeur (comme un nom d'utilisateur ou des informations d'identification de matériaux de clé publique) une chaîne GeneralizedTime, et un nombre aléatoire.

Le champ receiptsFrom est utilisé par le générateur pour spécifier les receveurs auxquels il est demandé de retourner un récépissé signé. Un CHOIX est fourni pour permettre de spécifier que :
- des récépissés sont demandés de tous les receveurs ;
- des récépissés de premier rang (receveurs qui n'ont pas reçu le message comme membres d'une liste de diffusion) des receveurs sont demandés ;
- des récépissés d'une liste de receveurs spécifique sont demandés.

```
ReceiptsFrom ::= CHOICE {
  allOrFirstTier [0] AllOrFirstTier,          -- anciennement "allOrNone [0]AllOrNone"
  receiptList [1] SEQUENCE OF GeneralNames }

AllOrFirstTier ::= INTEGER {                  -- anciennement AllOrNone
  allReceipts (0),
  firstTierRecipients (1) }
```

Le champ receiptsTo est utilisé par le générateur pour identifier le ou les utilisateurs auxquels le receveur identifié devrait envoyer des récépissés signés. Le générateur du message DOIT remplir le champ receiptsTo avec un GeneralNames pour chaque entité à qui le receveur devrait envoyeur le récépissé signé. Si le générateur du message veut que le receveur lui envoie le récépissé signé, il DOIT inclure un GeneralNames pour lui-même dans le champ receiptsTo.

2.8 Syntaxe du récépissé

Les récipissés sont représentés en utilisant un nouveau type de contenu, Receipt. Le type de contenu Receipt devra avoir le type ASN.1 Receipt. Les récépissés doivent être encapsulés au sein d'un message SignedData.

```
Receipt ::= SEQUENCE {
  version ESSVersion,
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }
```

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

ESSVersion ::= ENTIER { v1(1) }

Le champ Version définit le numéro de version de la syntaxe, qui est 1 pour cette version de la norme.

2.9 Indications de contenu

De nombreuses applications trouvent utile d'avoir des informations qui décrivent le contenu signé le plus interne d'un message multi couches disponibles sur la couche de signature la plus externe. L'attribut contentHints donne de telles informations.

Les valeurs de l'attribut Content-hints ont le type ASN.1 contentHints.

ContentHints ::= SEQUENCE { contentDescription UTF8String (SIZE (1..MAX)) OPTIONAL, contentType ContentType }

IDENTIFIANT D'OBJET id-aa-contentHint ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 4}

Le champ contentDescription peut être utilisé pour fournir des informations que le receveur peut utiliser pour choisir les messages protégés à traiter, comme l'objet d'un message. Si ce champ est établi, l'attribut est alors supposé apparaître sur l'objet signedData qui inclut un objet envelopedData et non sur l'objet signedData interne. La construction (SIZE (1..MAX)) contraint la séquence a avoir au moins une entrée. MAX indique que la limite supérieure est inspécifiée. Les mises en œuvre sont libres de choisir une limite supérieure qui convient à leur environnement.

Les messages qui contiennent un objet signedData enveloppé autour d'un objet envelopedData, masquant ainsi le type de contenu interne du message, DEVRAIENT inclure un attribut contentHints, sauf pour le cas du type de contenu Data. Des types de contenu de message spécifiques peuvent forcer ou empêcher l'inclusion de l'attribut contentHints. Par exemple, lorsque un signedData/Receipt est chiffré au sein d'un objet envelopedData, un objet signedData externe DOIT être créé qui encapsule l'objet envelopedData et un attribut contentHints avec le contentType réglé à l'identifiant d'objet id-ct-receipt DOIT être inclus dans le signedData SignerInfo signedAttributes externe.

2.10 Attribut Résumé de signature de message

L'attribut msgSigDigest ne peut être utilisé que dans les attributs signés d'un récépissé signé. Il contient le résumé des signedAttributes codés en DER ASN.1 inclus dans le signedData original qui demandait le récépissé signé. Un seul attribut msgSigDigest peut apparaître dans un ensemble d'attributs signés. Il est défini comme suit :

msgSigDigest ::= CHAINE D'OCTETS

IDENTIFIANT D'OBJET id-aa-msgSigDigest ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 5}

2.11 Attribut Référence de contenu signé

L'attribut contentReference est un lien d'un SignedData à un autre. Il peut être utilisé pour lier une réponse au message d'origine auquel elle se réfère, ou pour incorporer par référence un SignedData dans un autre. Le premier SignedData DOIT inclure un attribut contentIdentifier signé, qui DEVRAIT être construit comme spécifié au paragraphe 2.7. Le second SignedData se relie au premier par l'inclusion d'un attribut ContentReference signé qui contient le type de contenu, l'identifiant de contenu, et la valeur de signature provenant du premier SignedData.

```
ContentReference ::= SEQUENCE {
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue CHAINE D'OCTETS }
```

IDENTIFIANT D'OBJET id-aa-contentReference ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 10 }

3. Étiquettes de sécurité

Cette section décrit la syntaxe à utiliser pour les étiquettes de sécurité qui peuvent facultativement être associées aux données S/MIME encapsulées. Une étiquette de sécurité est un ensemble d'informations de sécurité concernant la sensibilité du contenu qui est protégé par l'encapsulation S/MIME.

L'"autorisation" est l'acte d'accorder des droits et/ou privilèges aux utilisateurs leur permettant d'accéder à un objet. Le "contrôle d'accès" est un moyen de mettre en application ces autorisations. Les informations de sensibilité dans une étiquette de sécurité peuvent être comparées avec les autorisations d'un usager pour déterminer si l'usager est autorisé à accéder au contenu qui est protégé par l'encapsulation S/MIME.

Les étiquettes de sécurité peuvent être utilisées pour d'autres objets comme une source d'informations d'acheminement. Les étiquettes décrivent souvent des niveaux ordonnés ("secret", "confidentiel", "restreint", et ainsi de suite) ou qui sont fondés sur le rôle, décrivant quelle sorte de personnes peuvent voir les informations ("équipe de santé du patient", "agents de facturation médicaux", "sans restriction", et ainsi de suite).

3.1 Règles de traitement des étiquettes de sécurité

Un agent envoyeur peut inclure un attribut d'étiquette de sécurité dans les attributs signés d'un objet signedData. Un agent receveur examine l'étiquette de sécurité sur un message reçu et détermine si le receveur est admis ou non à voir le contenu du message.

3.1.1 Ajout d'étiquettes de sécurité

Un agent envoyeur qui utilise une étiquette de sécurités DOIT mettre l'attribut étiquette de sécurité dans le champ signedAttributes d'un bloc SignerInfo. L'attribut étiquette de sécurité NE DOIT PAS être inclus dans des attributs non signés. Les services de sécurité d'intégrité et d'authentification DOIVENT être appliqués à l'étiquette de sécurité, donc elle DOIT être incluse dans un attribut signé, si elle est utilisée. C'est pour cela que l'attribut étiquette de sécurité fait partie des données qui sont hachées pour former la valeur de signature SignerInfo. Un bloc SignerInfo NE DOIT PAS avoir plus d'un attribut étiquette de sécurité signé.

Lorsque plusieurs blocs SignedData sont appliqués à un message, un attribut d'étiquette de sécurité peut être inclus dans l'une ou l'autre de la signature interne, de la signature externe, ou des deux. Un attribut étiquette de sécurité signé peut être inclus dans un champ signedAttributes au sein du bloc SignedData interne. L'étiquette de sécurité interne va inclure les sensibilités du contenu original et va être utilisée pour les décisions de contrôle d'accès en rapport avec le contenu encapsulé en texte source. La signature interne assure l'authentification de l'étiquette de sécurité interne et protège cryptographiquement l'étiquette de sécurité interne du contenu original du signataire original.

Lorsque le générateur signe le contenu et les attributs signés du texte source, l'étiquette de sécurité interne est liée au contenu du texte source. Une entité intermédiaire ne peut pas changer l'étiquette de sécurité interne sans invalider la signature interne. Le service de sécurité de confidentialité peut être appliqué à l'étiquette de sécurité interne en chiffrant l'objet signedData interne entier au sein d'un bloc EnvelopedData.

Un attribut étiquette de sécurité signé peut aussi être inclus dans un champ signedAttributes au sein du bloc SignedData externe. L'étiquette de sécurité externe va inclure les sensibilités du message chiffré et va être utilisée pour les décisions de contrôle d'accès en rapport avec le message chiffré et pour les décisions d'acheminement. La signature externe assure l'authentification de l'étiquette de sécurité externe (ainsi que du contenu encapsulé qui peut inclure des messages S/MIME incorporés).

Il peut y avoir plusieurs SignerInfos au sein d'un objet SignedData, et chaque SignerInfo peut inclure des signedAttributes. Donc, un seul objet SignedData peut inclure plusieurs eSSSecurityLabels, chaque SignerInfo ayant un attribut eSSSecurityLabel. Par exemple, l'origine peut envoyer un message signé avec deux SignerInfo, l'un contenant une signature DSS, l'autre contenant une signature RSA. Si un des SignerInfo inclus dans un objet SignedData comporte un attribut eSSSecurityLabel, tous les SignerInfo dans cet objet SignedData DOIVENT alors inclure un attribut eSSSecurityLabel et la valeur de chacun DOIT être identique.

3.1.2 Traitement des étiquettes de sécurité

Avant de traiter un signedAttribute eSSSecurityLabel, l'agent receveur DOIT vérifier la signature des SignerInfo qui couvrent l'attribut eSSSecurityLabel. Un receveur NE DOIT PAS traiter un attribut eSSSecurityLabel qui n'a pas été vérifié.

Un agent receveur DOIT traiter l'attribut eSSSecurityLabel, si présent, dans chaque SignerInfo dans l'objet SignedData pour lequel il vérifie la signature. Il peut en résulter que l'agent receveur traite plusieurs eSSSecurityLabel incluses dans un seul objet SignedData. Comme toutes les eSSSecurityLabel dans un objet SignedData doivent être identiques, l'agent receveur traite (comme on effectue un contrôle d'accès) la première eSSSecurityLabel qu'il rencontre dans une SignerInfo qu'il vérifie, et s'assure ensuite que toutes les autres eSSSecurityLabel dans les signerInfo qu'il vérifie sont identiques à la première rencontrée. Si les eSSSecurityLabel dans les signerInfo qu'il vérifie ne sont pas toutes identiques, l'agent receveur DOIT avertir l'utilisateur de cette condition.

Les agents receveurs DEVRAIENT avoir une politique locale pour ce qui concerne l'exposition du contenu interne d'un objet signedData qui inclut un security-policy-identifier eSSSecurityLabel que le logiciel de traitement ne reconnaît pas. Si l'agent receveur ne reconnaît pas la valeur de l'identifiant de politique de sécurité eSSSecurityLabel, il DEVRAIT alors arrêter le traitement du message et indiquer une erreur.

3.2 Syntaxe de eSSSecurityLabel

La syntaxe de eSSSecurityLabel dérive directement du module ASN.1 de [X.411]. (Le module MTSAbstractService commence par "DEFINITIONS IMPLICIT TAGS ::=".) De plus, la syntaxe de eSSSecurityLabel est compatible avec celle utilisée dans [MSP4].

```
ESSSecurityLabel ::= SET {
  security-policy-identifier SecurityPolicyIdentifier,
  security-classification SecurityClassification OPTIONAL,
  privacy-mark ESSPrivacyMark OPTIONAL,
  security-categories SecurityCategories OPTIONAL }

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

SecurityPolicyIdentifier ::= IDENTIFIANT D'OBJET

SecurityClassification ::= ENTIER {
  unmarked (0),
  unclassified (1),
  restricted (2),
  confidential (3),
  secret (4),
  top-secret (5) } (0..ub-integer-options)

ub-integer-options ENTIER ::= 256

ESSPrivacyMark ::= CHOIX {
   pString      PrintableString (SIZE (1..ub-privacy-mark-length)),
   utf8String   UTF8String (SIZE (1..MAX))
}

ub-privacy-mark-length ENTIER ::= 128

SecurityCategories ::= SET SIZE (1..ub-security-categories) OF SecurityCategory

ub-security-categories ENTIER ::= 64

SecurityCategory ::= SEQUENCE {
  type  [0] IDENTIFIANT D'OBJET,
  value [1] ANY DEFINED BY type                    – défini par type
}
```

Note : la syntaxe susmentionnée de SecurityCategory produit des codages hexadécimaux identiques à la syntaxe suivante de SecurityCategory qui est documentée dans la spécification X.411 :

```
--SecurityCategory ::= SEQUENCE {
--     type  [0]  SECURITY-CATEGORY,
```

```
--     value [1]  ANY DEFINED BY type }
--
--SECURITY-CATEGORY MACRO ::=
--BEGIN
--TYPE NOTATION ::= type | empty
--VALUE NOTATION ::= value (VALEUR D'IDENTIFIANT D'OBJET)
--FIN
```

3.3 Composants de l'étiquette de sécurité

Ce paragraphe donne des détails sur les divers composants de la syntaxe de eSSSecurityLabel.

3.3.1 Identifiant de politique de sécurité

Une politique de sécurité est un ensemble de critères pour la fourniture de services de sécurité. L'identifiant de politique de sécurité eSSSecurityLabel est utilisé pour identifier la politique de sécurité en cours à laquelle l'étiquette de sécurité se rapporte. Il indique la sémantique des autres composants d'étiquette de sécurité.

3.3.2 Classification de la sécurité

La présente spécification définit l'utilisation du champ Classification de sécurité exactement comme spécifiée dans la Recommandation X.411, qui déclare entre autres :

Si présente, une classification de sécurité peut avoir une des valeurs d'une liste hiérarchisée. La hiérarchie de base de classification de sécurité est définie dans cette Recommandation, mais l'utilisation de ces valeurs est définie par la politique de sécurité en vigueur. Des valeurs supplémentaires de classification de sécurité, et leur position dans la hiérarchie, peuvent aussi être définies par une politique de sécurité comme une affaire locale ou par accord bilatéral. La hiérarchie de base de classification de sécurité est, en ordre ascendant : non marqué, non classifié, restreint, confidentiel, secret, très secret.

Cela signifie que la politique de sécurité en vigueur (identifiée par l'identifiant de politique de sécurité eSSSecurityLabel) définit des valeurs entières de SecurityClassification et leur signification.

Une organisation peut développer sa propre politique de sécurité qui définit les valeurs d'ENTIER de SecurityClassification et leur signification. Cependant, l'interprétation générale de la spécification X.411 est que les valeurs de 0 à 5 sont réservées pour les valeurs de la "hiérarchie de base" de non marqué, non classifié, restreint, confidentiel, secret, et très secret. Noter que X.411 ne fournit pas les règles d'utilisation de ces valeurs pour étiqueter les données et comment le contrôle d'accès est effectué en utilisant ces valeurs.

Il n'y a pas de définition universelle des règles d'utilisation des valeurs de cette "hiérarchie de base". Chaque organisation (ou groupe d'organisations) va définir une politique de sécurité qui documente comment les valeurs de la "hiérarchie de base" sont utilisées (si elles le sont) et comment le contrôle d'accès est appliqué (si il l'est) au sien de son domaine.

Donc, la valeur de la classification de sécurité DOIT être accompagnée par une valeur d'identifiant de politique de sécurité pour définir les règles de son utilisation. Par exemple, la classification "secret" d'une entreprise peut porter une signification différente de celle du Gouvernement des USA. En résumé, une politique de sécurité NE DEVRAIT PAS utiliser les entiers 0 à 5 pour d'autres significations que celles de X.411, et DEVRAIENT à la place utiliser d'autres valeurs hiérarchisées.

Noter que l'ensemble des valeurs de classification de sécurité valides DOIT être hiérarchique, mais ces valeurs n'ont pas nécessairement besoin d'être en ordre numérique ascendant. De plus, les valeurs n'ont pas besoin d'être contiguës.

Par exemple, dans la politique de sécurité du système de message n° 1.0 de la défense, la valeur 11 de la classification de sécurité indique "sensible mais non classifié" et 5 indiques "très secret". La hiérarchie des sensibilité range "très secret" comme plus sensible que "sensible mais non classifié" bien que la valeur numérique de "très secret" soit inférieure à celle de "sensible mais non classifié".

(Bien sûr, si les valeurs de la classification de sécurité sont à la fois hiérarchisées et en ordre ascendant, un lecteur occasionnel de la politique de sécurité a plus de chances de la comprendre.)

Un exemple d'une politique de sécurité qui n'utilise aucune des valeurs de X.411 pourrait être :
10 -- tous
15 -- Morgan Corporation et ses co-contractants

20 – employés de Morgan Corporation
25 – membres du conseil des directeurs de Morgan Corporation

Un exemple d'une politique de sécurité qui utilise en partie la hiérarchie de X.411 pourrait être :
0 – non marqué
1 – non classifié, peut être lu par tous
2 – restreint au personnel de Timberwolf Productions
6 – ne peut être lu que par les dirigeants de Timberwolf Productions.

3.3.3 Marque de confidentialité

Si elle est présente, la marque de confidentialité eSSSecurityLabel n'est pas utilisée pour le contrôle d'accès. Le contenu de la marque de confidentialité eSSSecurityLabel peut être défini par la politique de sécurité en vigueur (identifiée par l'identifiant de politique de sécurité eSSSecurityLabel) qui peut définir une liste de valeurs à utiliser. Autrement, la valeur peut être déterminée par l'origine de l'étiquette de sécurité.

3.3.4 Catégories de sécurité

Si elle est présente, la catégorie de sécurité eSSSecurityLabel fournit plus de granularité pour la sensibilité du message. La politique de sécurité en vigueur (identifiée par l'identifiant de politique de sécurité eSSSecurityLabel) est utilisée pour indiquer les syntaxes permises dans les catégories de sécurité eSSSecurityLabel. Autrement, les catégories de sécurité et leurs valeurs peuvent être définies par accord bilatéral.

3.4 Étiquettes de sécurité équivalentes

Parce qu'il est permis aux organisations de définir leurs propres politiques de sécurité, de nombreuses politiques de sécurité différentes vont exister. Certaines organisations peuvent souhaiter créer des équivalences entre leurs politiques de sécurité et celles d'autres organisations. Par exemple, la Compagnie Acme et la Widget Corporation peuvent passer un accord bilatéral selon lequel la valeur de classification de sécurité "Acme private" est équivalente à la valeur de classification de sécurité "Widget sensitive".

Les agents receveurs NE DOIVENT PAS traiter un attribut equivalentLabels dans un message si l'agent ne fait pas confiance au signataire de cet attribut pour traduire les valeurs de eSSSecurityLabel originales dans la politique de sécurité incluse dans l'attribut equivalentLabels. Les agents receveurs ont l'option de traiter l'attribut equivalentLabels mais ne sont pas obligés de le faire. Il est acceptable qu'un agent receveur ne traite que les eSSSecurityLabels. Tous les agents receveurs DEVRAIENT reconnaître les attributs equivalentLabels même si ils ne les traitent pas.

3.4.1 Création d'étiquettes équivalentes

L'attribut signé EquivalentLabels est défini par :

EquivalentLabels ::= SEQUENCE DE ESSSecurityLabel

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

Comme mentionné précédemment, une ESSSecurityLabel contient les valeurs de sensibilité choisies par le signataire original des signedData. Si une ESSSecurityLabel est présente dans des signerInfo, toutes les signerInfo dans les signedData DOIVENT contenir une ESSSecurityLabel et elles DOIVENT être toutes identiques. En plus d'une ESSSecurityLabel, un signerInfo PEUT aussi inclure un attribut signé equivalentLabels. Si il est présent, l'attribut equivalentLabels DOIT inclure une ou plusieurs étiquettes de sécurité qui sont estimées par le signataire être sémantiquement équivalentes à l'attribut ESSSecurityLabel inclus dans le même signerInfo.

Tous les identifiants d'objet de politique de sécurité DOIVENT être uniques dans l'ensemble des étiquettes de sécurité ESSSecurityLabel et EquivalentLabels. Avant d'utiliser un attribut EquivalentLabels, un agent receveur DOIT s'assurer que tous les OID de politique de sécurité sont uniques dans les étiquettes de sécurité ou étiquettes incluses dans les EquivalentLabels. Une fois que l'agent receveur a choisi l'étiquette de sécurité (parmi les EquivalentLabels) à utiliser pour le traitement, l'OID de politique de sécurité des étiquettes de sécurité EquivalentLabels choisies DOIT être comparé à celui de ESSSecurityLabel pour s'assurer qu'il est unique.

Dans le cas où un attribut ESSSecurityLabel n'est pas inclus dans un signerInfo, un attribut EquivalentLabels peut encore être inclus. Par exemple, dans la politique de sécurité de Acme, l'absence d'une ESSSecurityLabel pourrait être définie comme égale à une étiquette de sécurité composée de l'OID de politique de sécurité de Acme et de la classification de sécurité "non marqué".

Noter que les equivalentLabels NE DOIVENT PAS être utilisées pour convoyer des étiquettes de sécurité qui sont sémantiquement différentes de la ESSSecurityLabel incluse dans les signerInfo dans les signedData. Si une entité a besoin d'appliquer une étiquette de sécurité qui est sémantiquement différente de la ESSSecurityLabel, elle DOIT alors inclure l'étiquette de sécurité sémantiquement différente dans un objet signedData externe qui encapsule l'objet signedData qui inclut la ESSSecurityLabel.

Si il est présent, l'attribut equivalentLabels DOIT être un attribut signé ; il NE DOIT PAS être un attribut non signé. La [RFC2630] définit signedAttributes comme un ENSEMBLE DE attributs. Une signerInfo NE DOIT PAS inclure plusieurs instances de l'attribut equivalentLabels. La CMS définit la syntaxe ASN.1 pour les attributs signés comme incluant un ENSEMBLE DE attrValues de AttributeValue. Un attribut equivalentLabels ne DOIT inclure qu'une seule instance de AttributeValue. Il NE DOIT PAS y avoir zéro ou plusieurs instances de AttributeValue présentes dans l'ENSEMBLE DE attrValues de AttributeValue.

3.4.2 Traitement d'étiquettes équivalentes

Un agent receveur DEVRAIT traiter la ESSSecurityLabel avant de traiter aucune EquivalentLabels. Si la politique dans la ESSSecurityLabel est comprise par l'agent receveur, il DOIT traiter cette étiquette et DOIT ignorer toutes les EquivalentLabels.

Lorsque il traite un attribut EquivalentLabels, l'agent receveur DOIT valider la signature sur l'attribut EquivalentLabels. Un agent receveur NE DOIT PAS agir sur un attribut equivalentLabels pour lequel la signature ne pourrait pas être validée, et NE DOIT PAS agir sur un attribut equivalentLabels si cet attribut n'est pas signé par une entité de confiance pour traduire les valeurs de la eSSSecurityLabel originale dans la politique de sécurité incluse dans l'attribut equivalentLabels. Déterminer à qui il est permis de spécifier les transpositions d'équivalence est une politique locale. Si un message a plus d'un attribut EquivalentLabels, l'agent receveur DEVRAIT traiter le premier qu'il lit et valider ce qu'il contient sur la politique de sécurité qui concerne l'agent receveur.

4. Gestion des listes de diffusion

Les agents envoyeurs doivent créer des structures de données spécifiques du receveur pour chaque receveur d'un message chiffré. Ce processus peut dégrader les performances pour les messages envoyés à un grand nombre de receveurs. Donc, des agents de liste de diffusion (MLA) qui peuvent prendre un seul message et effectuer le chiffrement spécifique du receveur pour chaque receveur sont souvent souhaités.

Un MLA apparaît au générateur du message comme un receveur normal du message, mais le MLA agit comme un point d'expansion de message pour une liste de diffusion (ML, *Mail List*). L'envoyeur d'un message dirige le message sur le MLA, qui redistribue alors le message aux membres de la liste de diffusion. Ce procès décharge du traitement par receveur les agents d'utilisateur individuels et permet une gestion plus efficace des grandes listes de diffusion. Les listes de diffusion sont de vrais receveurs de message desservis par les MLA qui fournissent les services de cryptographie et d'expansion pour la liste de diffusion.
En plus du traitement cryptographique des messages, des listes de diffusion sûres ont aussi à prévenir les boucles de messagerie. Une boucle de messagerie se produit lorsque une liste de diffusion est membre d'une autre liste de diffusion, et que la seconde liste de diffusion est membre de la première. Un message va aller d'une liste à l'autre dans une succession en cascade de messages qui seront distribués à tous les autres membres des deux listes.

Pour empêcher les boucles de messagerie, les MLA utilisent l'attribut mlExpansionHistory de la signature externe d'un message à triple enveloppe. L'attribut mlExpansionHistory est essentiellement une liste de chaque MLA qui a traité le message. Si un MLA voit son propre identifiant unique d'entité dans la liste, il sait qu'une boucle a été formée, et n'envoie pas à nouveau le message à la liste.

4.1 Expansion de liste de diffusion

Le traitement de l'expansion de la liste de diffusion est noté dans la valeur de l'attribut mlExpansionHistory, situé dans les attributs signés du bloc SignerInfo du MLA. Le MLA crée ou met à jour la valeur de l'attribut mlExpansionHistory signé

chaque fois que le MLA répand et signe un message pour les membres d'une liste de diffusion.

Le MLA DOIT ajouter un enregistrement MLData contenant les informations d'identification du MLA, la date et l'heure d'expansion, et la politique facultative de réception à la fin de la séquence d'historique d'expansion de la liste de diffusion. Si l'attribut mlExpansionHistory est absent, le MLA DOIT alors ajouter l'attribut et l'expansion en cours devient le premier élément de la séquence. Si l'attribut mlExpansionHistory est présent, le MLA DOIT ajouter les informations d'expansion en cours à la fin de la séquence existante de MLExpansionHistory. Un seul attribut mlExpansionHistory peut être inclus dans le signedAttributes d'une SignerInfo.

Noter que si l'attribut mlExpansionHistory est absent, le receveur est alors un receveur de message de premier rang.

Il peut y avoir plusieurs SignerInfo au sein d'un objet SignedData, et chaque SignerInfo peut inclure des attributs signés. Donc, un seul objet SignedData peut inclure plusieurs SignerInfo, chaque SignerInfo ayant un attribut mlExpansionHistory. Par exemple, un MLA peut envoyer un message signé avec deux SignerInfo, l'un contenant une signature DSS, l'autre contenant une signature RSA.

Si un MLA crée un SignerInfo qui inclut un attribut mlExpansionHistory, tous les SignerInfo créés par le MLA pour cet objet SignedData DOIVENT alors inclure un attribut mlExpansionHistory, et la valeur de chacun DOIT être identique. Noter que d'autres agents peuvent ultérieurement ajouter des attributs SignerInfo au bloc SignedData, et ces SignerInfo supplémentaires pourraient ne pas inclure d'attribut mlExpansionHistory.

Un receveur DOIT vérifier la signature du signerInfo qui couvre l'attribut mlExpansionHistory avant de traiter le mlExpansionHistory, et NE DOIT PAS traiter l'attribut mlExpansionHistory tant que sa signature n'a pas été vérifiée. Si un objet SignedData a plus d'un signerInfo qui a un attribut mlExpansionHistory, le receveur DOIT comparer les attributs mlExpansionHistory dans tous les signerInfo qu'il a vérifié, et NE DOIT PAS traiter l'attribut mlExpansionHistory tant qu'il n'a pas vérifié que tous les attributs mlExpansionHistory dans le bloc SignedData sont identiques. Si les attributs mlExpansionHistory dans les signerInfo vérifiés ne sont pas tous identiques, l'agent receveur DOIT arrêter le traitement du message et DEVRAIT notifier à l'utilisateur ou à l'administrateur du MLA cette condition d'erreur. Dans le traitement de mlExpansionHistory, les signerInfo qui n'ont pas d'attribut mlExpansionHistory sont ignorés.

4.1.1 Détection des boucles d'expansion de liste de diffusion

Avant de développer un message, le MLA examine la valeur de tout attribut d'historique d'expansion de liste de diffusion existant pour détecter une boucle d'expansion. Une boucle d'expansion existe lorsque un message déployé par un MLA spécifique pour une liste de diffusion spécifique est délivré à nouveau au même MLA pour la même liste de diffusion.

Les boucles d'expansion sont détectées en examinant le champ mailListIdentifier de chaque entrée MLData trouvée dans l'historique d'expansion de la liste de diffusion. Si un MLA trouve ses propres informations d'identification, il doit alors interrompre le processus d'expansion et devrait fournir un avertissement de boucle d'expansion à un administrateur humain de la liste de diffusion. L'administrateur de la liste de diffusion est responsable de la correction de la condition de boucle.

4.2 Traitement par l'agent de liste de messagerie

Les premiers alinéas de ce paragraphe donnent une description générale du traitement d'un MLA. Le reste du paragraphe en donne une description détaillée.

Le traitement du message par le MLA dépend de la structure des couches S/MIME dans le message envoyé au MLA pour expansion. En plus d'envoyer le message à triple enveloppes à un MLA, une entité peut envoyer d'autres types de messages à un MLA, comme :

- un seul message signedData enveloppé ou envelopedData,
- un message à double enveloppe (comme signé et enveloppé, enveloppé et signé, ou signé et signé, et ainsi de suite)
- un message à quadruple enveloppe (comme si un message à triple enveloppe bien formé était envoyé à travers une passerelle qui y ajoute un couche SignedData externe).

Dans tous les cas, le MLA DOIT analyser toutes les couches du message reçu pour déterminer si il y a des couches signedData qui comportent un signedAttribute eSSSecurityLabel. Cela peut inclure de déchiffrer une couche EnvelopedData pour déterminer si une couche SignedData encapsulée comporte un attribut eSSSecurityLabel. Le MLA DOIT traiter complètement chaque attribut eSSSecurityLabel trouvé dans les diverses couches de signedData, incluant d'effectuer les vérifications de contrôle d'accès, avant de distribuer le message aux membres de la liste de diffusion. Les détails des vérifications de contrôle d'accès sortent du domaine d'application du présent document. Le MLA DOIT vérifier la signature du signerInfo incluant l'attribut eSSSecurityLabel avant de l'utiliser.

Dans tous les cas, le MLA DOIT signer le message à envoyer aux membres de la liste de diffusion dans une nouvelle couche signedData "externe". Le MLA DOIT ajouter ou mettre à jour un attribut mlExpansionHistory dans les signedData "externes" qu'il crée pour documenter le traitement du MLA. Si il y avait une couche signedData "externe" incluse dans le message original reçu par le MLA, la couche de signedData "externe" créée par le MLA DOIT alors inclure chaque attribut signé présent dans la couche de signedData "externe" d'origine, sauf si le MLA remplace explicitement un attribut (comme un signingTime ou mlExpansionHistory) par une nouvelle valeur.

Lorsque un message S/MIME est reçu par le MLA, celui-ci DOIT d'abord déterminer quelle couche signedData reçue, si il en est, est la couche signedData "externe". Pour identifier la couche signedData "externe" reçue, le MLA DOIT vérifier la signature et traiter complètement les signedAttributes dans chacune des couches signedData externes (en travaillant de l'extérieur vers l'intérieur) pour déterminer si l'un ou l'autre d'entre eux inclut un attribut mlExpansionHistory ou encapsule un objet envelopedData.

La recherche du MLA de la couche signedData "externe" est terminée lorsque il trouve un de ce qui suit :
- la couche signedData "externe" qui comporte un attribut mlExpansionHistory ou encapsule un objet envelopedData,
- une couche envelopedData,
- le contenu original (c'est-à-dire, une couche qui n'est ni envelopedData ni signedData).

Si le MLA trouve une couche signedData "externe", il DOIT alors effectuer les étapes suivantes :
1. retirer toutes les couches de signedData qui encapsulaient la couche signedData "externe",
2. retirer la couche signedData "externe" elle-même (après avoir mémorisé les signedAttributes inclus)
3. développer les envelopedData (si présentes)
4. signer le message à envoyer aux membres de la liste de diffusion dans une nouvelle couche signedData "externe" qui comporte les signedAttributes (sauf explicitement remplacés) de la couche originale de signedData "externe".

Si le MLA trouve une couche signedData "externe" qui inclut un attribut mlExpansionHistory ET si le MLA trouve ensuite une couche envelopedData insérée plus profondément dans les couches du message reçu, le MLA DOIT alors supprimer toutes les couches de signedData jusqu'à la couche envelopedData (incluant de supprimer la couche originale de signedData "externe") et DOIT signer les envelopedData expansées dans une nouvelle couche signedData "externe" qui inclut les signedAttributes (sauf explicitement remplacés) provenant de la couche originale de signedData "externe" reçue.

Si le MLA ne trouve pas une couche signedData "externe" ET ne trouve pas une couche envelopedData, le MLA DOIT alors signer le message original reçu dans une nouvelle couche de signedData "externe". Si le MLA ne trouve pas une signedData "externe" ET ne trouve pas une couche envelopedData, il DOIT alors expanser la couche envelopedData, si elle est présente, et la signer dans une nouvelle couche signedData "externe".

4.2.1 Exemples de traitement de règle

Les exemples suivants aident à expliquer les règles ci-dessus :

1) Un message (S1(Contenu original)) (où S = SignedData) est envoyé au MLA dans lequel la couche signedData ne comporte pas d'attribut MLExpansionHistory. Le MLA vérifie et traite complètement les signedAttributes en S1. Le MLA décide qu'il n'y a pas de couche signedData originale "externe" reçue puisqu'il trouve le contenu original, mais pas d'attribut envelopedData ni mlExpansionHistory. Le MLA calcule une nouvelle couche signedData, S2, résultant en le message suivant envoyé aux receveurs de la liste de diffusion : (S2(S1(Contenu original))). Le MLA inclut un attribut mlExpansionHistory en S2.

2) Un message (S3(S2(S1(Contenu original)))) est envoyé au MLA dans lequel aucune des couches signedData n'inclut un attribut MLExpansionHistory. Le MLA vérifie et traite complètement les signedAttributes dans S3, S2 et S1. Le MLA décide qu'il n'y a pas de couche signedData originale "externe" reçue puisqu'il trouve le contenu original, mais pas d'attribut envelopedData ni mlExpansionHistory. Le MLA calcule une nouvelle couche signedData, S4, résultant en l'envoi du message suivant envoyé aux receveurs de la liste de diffusion : (S4(S3(S2(S1(Contenu original))))). Le MLA inclut un attribut mlExpansionHistory dans S4.

3) Un message (E1(S1(Contenu original))) (où E = envelopedData) est envoyé au MLA dans lequel S1 n'inclut pas d'attribut MLExpansionHistory. Le MLA décide qu'il n'y a pas de couche signedData originale "externe" reçue puisqu'il trouve E1 comme couche externe. Le MLA fait l'expansion de recipientInformation dans E1. Le MLA calcule une nouvelle couche signedData, S2, résultant en l'envoi du message suivant aux receveurs de la liste de diffusion : (S2(E1(S1(Contenu original)))). Le MLA inclut un attribut mlExpansionHistory dans S2.

4) Un message (S2(E1(S1(Contenu original)))) est envoyé au MLA, dans lequel S2 inclut un attribut MLExpansionHistory.

Le MLA vérifie la signature et traite complètement les signedAttributes dans S2. Le MLA trouve l'attribut mlExpansionHistory dans S2, et décide donc que S2 est le signedData "externe". Le MLA mémorise les signedAttributes inclus dans S2 pour les inclure ensuite dans les nouvelles signedData externes qu'il applique au message. Le MLA supprime S2. Le MLA fait ensuite l'expansion des recipientInformation dans E1 (cela invalide la signature dans S2 et c'est pour cela qu'il a été supprimé). Le MLA calcule une nouvelle couche signedData, S3, résultent dans le message suivant envoyé aux receveurs de la liste de diffusion : (S3(E1(S1(Contenu original)))). Le MLA inclut dans S3 les attributs provenant de S2 (sauf si il remplace spécifiquement une valeur d'attribut) incluant un attribut mlExpansionHistory mis à jour.

5) Un message (S3(S2(E1(S1(Contenu original))))) est envoyé au MLA dans lequel aucune couche signedData n'inclut un attribut MLExpansionHistory. Le MLA vérifie la signature et traite complètement les signedAttributes dans S3 et S2. Lorsque le MLA rencontre E1, il décide alors que S2 est la signedData "externe" car S2 encapsule E1. Le MLA mémorise les signedAttributes inclus dans S2 pour les inclure ultérieurement dans la nouvelle signedData externe qu'il applique au message. Le MLA supprime S3 et S2. Le MLA procède alors à l'expansion des recipientInformation dans E1 (cela invalide les signatures dans S3 et S2 et c'est pour cela qu'ils ont été supprimés). Le MLA calcule une nouvelle couche signedData, S4, résultant en le message suivant envoyé aux receveurs de la liste de diffusion : (S4(E1(S1(Contenu original)))). Le MLA inclut dans S4 les attributs provenant de S2 (sauf si il remplace spécifiquement une valeur d'attribut) et inclut un nouvel attribut mlExpansionHistory.

6) Un message (S3(S2(E1(S1(Contenu original))))) est envoyé au MLA dans lequel S3 comporte un attribut MLExpansionHistory. Dans ce cas, le MLA vérifie la signature et traite complètement les signedAttributes dans S3. Le MLA trouve le mlExpansionHistory dans S3, et décide donc que S3 est le signedData "externe". Le MLA mémorise les signedAttributes inclus dans S3 pour les inclure ultérieurement dans le nouvel signedData externe qu'il applique au message. Le MLA continue d'analyser les couches encapsulées parce qu'il doit déterminer si elles contiennent des attributs eSSSecurityLabel. Le MLA vérifie la signature et traite complètement les signedAttributes dans S2. Lorsque le MLA rencontre E1, il supprime alors S3 et S2. Le MLA procède alors à l'expansion des recipientInformation dans E1 (cela invalide les signatures dans S3 et S2 ce qui est la raison de leur suppression). Le MLA calcule une nouvelle couche signedData, S4, ce qui résulte en l'envoi du message suivant aux receveurs de la liste de diffusion : (S4(E1(S1(Contenu original)))). Le MLA inclut dans S4 les attributs provenant de S3 (sauf si il remplace spécifiquement une valeur d'attribut) incluant un attribut mlExpansionHistory mis à jour.

4.2.3 Choix de traitements

Le traitement utilisé dépend du type de la couche la plus externe du message. Il y a trois cas de type des données externes :
- EnvelopedData
- SignedData
- data

4.2.3.1 Traitement de EnvelopedData

1. Le MLA localise ses propres RecipientInfo et utilise les informations qu'il contient pour obtenir la clé du message.

2. Le MLA retire le champ recipientInfos existant et le remplace par une nouvelle valeur de recipientInfos construite à partir des structures RecipientInfo créées pour chaque membre de la liste de diffusion. Le MLA retire aussi le champ originatorInfo existant et le remplace par une nouvelle valeur de originatorInfo construite à partir des informations qui décrivent le MLA.

3. Le MLA encapsule le message chiffré développé dans un bloc SignedData, ajoutant un attribut mlExpansionHistory comme décrit au paragraphe 4.1 "Expansion de liste de diffusion" pour documenter l'expansion.

4. Le MLA signe le nouveau message et livre le message mis à jour aux membres de la liste de diffusion pour achever le traitement par le MLA.

4.2.3.2 Traitement de SignedData

Le traitement par le MLA de messages multi couches dépend du type de données dans chacune des couches. L'étape 3 ci-dessous spécifie que des traitements différents vont avoir lieu selon le type de message de CMS qui a été signé. C'est-à-dire qu'il a besoin de savoir le type des données à la prochaine couche interne, qui peut ou non être la couche la plus interne.

1. Le MLA vérifie la valeur de signature trouvée dans la couche SignedData la plus interne associée aux données signées. Le traitement du message par le MLA se termine si la signature du message est invalide.

2. Si la couche SignedData la plus externe inclut un attribut mlExpansionHistory signé, le MLA vérifie qu'il n'y a pas une boucle d'expansion comme décrit au paragraphe 4.1.1 "Détection de boucles d'expansion de liste de diffusion", puis passe à l'étape 3. Si la couche SignedData la plus externe ne comporte pas d'attribut mlExpansionHistory signé, le MLA signe le message entier (incluant cette couche SignedData externe qui n'a pas d'attribut mlExpansionHistory), et livre le message mis à jour aux membres de la liste de diffusion pour achever le traitement du MLA.

3. Déterminer le type de données qui ont été signées. C'est-à-dire, regarder le type des données sur la couche juste en dessous des SignedData, qui peut être ou non la couche "la plus interne". Sur la base du type de données, effectuer soit l'étape 3.1 (EnvelopedData), soit l'étape 3.2 (SignedData), soit l'étape 3.3 (tous les autres types).
   3.1. Si les données signées sont EnvelopedData, le MLA effectue le traitement d'expansion du message chiffré comme décrit précédemment. Noter que ce processus invalide la valeur de signature dans la couche SignedData la plus externe associée au message chiffré original. Passer en 3.2 avec le résultat de l'expansion.
   3.2. Si les données signées sont SignedData, ou est le résultat de l'expansion d'un bloc EnvelopedData dans l'étape 3.1 :
      3.2.1. le MLA supprime la couche existante de SignedData la plus externe après avoir mémorisé la valeur du mlExpansionHistory et de tous les autres attributs signés dans cette couche, si présents ;
      3.2.2. si les données signées sont EnvelopedData (provenant de l'étape 3.1), le MLA encapsule les message chiffré après expansion dans une nouvelle couche SignedData la plus externe. D'un autre côté, si les données signées sont des SignedData (provenant de l'étape 3.2) le MLA encapsule les données signées dans une nouvelle couche SignedData la plus externe ;
      3.2.3. la couche de signedData la plus externe créée par le MLA remplace la couche de signedData la plus externe originale. Le MLA DOIT créer une liste d'attributs signés pour la nouvelle couche de signedData la plus externe qui DOIT inclure chaque attribut signé présent dans la couche signedData originale la plus externe, sauf si le MLA remplace explicitement un ou plusieurs attributs particuliers par une nouvelle valeur. Un cas particulier est celui où il s'agit de l'attribut mlExpansionHistory. Le MLA DOIT ajouter un attribut signé mlExpansionHistory à la couche signedData externe comme suit :
         3.2.3.1 si la couche de SignedData la plus externe d'origine comportait un attribut mlExpansionHistory, la valeur de l'attribut est copiée et mise à jour avec les informations actuelles d'expansion de liste de diffusion comme décrit au paragraphe 4.1 "Expansion de liste de diffusion" ;
         3.2.3.2 si la couche de SignedData la plus externe d'origine ne comportait pas d'attribut mlExpansionHistory, une nouvelle valeur d'attribut est créée avec les informations actuelles d'expansion de liste de diffusion comme décrit au paragraphe 4.1 "Expansion de liste de diffusion".
   3.3 Si les données signées ne sont ni EnvelopedData ni SignedData :
      3.3.1 le MLA encapsule l'objet signedData reçu dans un objet SignedData externe, et ajoute un attribut mlExpansionHistory à l'objet SignedData externe contenant les informations actuelles d'expansion de liste de diffusion comme décrit au paragraphe 4.1 "Expansion de liste de diffusion".

4. Le MLA signe le nouveau message et livre le message mis à jour aux membres de la liste de diffusion pour achever le traitement.

Un diagramme des étapes ci-dessus serait :

1. Y a t-il une signature valide ?
   OUI -> 2.
   NON -> STOP.
2. La couche SignedData la plus externe contient elle un mlExpansionHistory ?
   OUI -> le vérifier, puis -> 3.
   NON -> Signer le message (y compris le SignedData le plus externe qui n'a pas de mlExpansionHistory), le livrer, STOP.
3. Vérifier le type des données juste en dessous du SignedData le plus externe.
   EnvelopedData -> 3.1.
   SignedData -> 3.2.
   tous les autres -> 3.3.

3.1. Procéder à l'expansion du message chiffré, puis -> 3.2.

3.2. -> 3.2.1.

3.2.1. Supprimer la couche SignedData la plus externe, noter la valeur de mlExpansionHistory et des autres attributs signés, puis -> 3.2.2.

3.2.2. Encapsuler dans la nouvelle signature, puis -> 3.2.3.

3.2.3. Créer une nouvelle couche signedData. Y avait il une ancienne mlExpansionHistory ?
   OUI -> copier les valeurs de l'ancienne mlExpansionHistory, puis -> 4.
   NON -> créer une nouvelle valeur de mlExpansionHistory, puis -> 4.

3.3. L'encapsuler dans une couche SignedData et ajouter un attribut mlExpansionHistory, puis -> 4.

4. Signer le message, le livrer, STOP.

4.2.3.3 Traitement des données

1. Le MLA encapsule le message dans une couche SignedData, et ajoute un attribut mlExpansionHistory contenant les informations actuelles d'expansion de liste de diffusion comme décrit au paragraphe "Expansion de liste de diffusion".

2. Le MLA signe le nouveau message et livre le message mis à jour aux membres de la liste de diffusion pour achever le traitement.

4.3 Traitement de la politique de récépissé signé par l'agent de liste de diffusion

Si une liste de diffusion (B) est membre d'une autre liste de diffusion (A), la liste B a souvent besoin de répercuter la politique de réception de liste de diffusion de A. En règle générale, une liste de diffusion devrait être prudente dans la répercussion de la politique de réception de liste de diffusion parce que le dernier receveur a seulement besoin de traiter le dernier élément dans l'historique d'expansion de la liste de diffusion. Le MLA construit l'historique d'expansion pour satisfaire cette exigence.

Le tableau ci-dessous décrit le résultat de l'union de la politique de liste de diffusion de A (les rangées du tableau) et de celle de B (les colonnes du tableau).

Politique de A	**Politique de B**			
	aucun	insteadOf	inAdditionTo	manquant
aucun	aucun	aucun	aucun	aucun
insteadOf	aucun	insteadOf(B)	*1	insteadOf(A)
inAdditionTo	aucun	insteadOf(B)	*2	inAdditionTo(A)
manquant	aucun	insteadOf(B)	inAdditionTo(B)	manquant

*1 = insteadOf(insteadOf(A) + inAdditionTo(B))
*2 = inAdditionTo(inAdditionTo(A) + inAdditionTo(B))

4.4 Syntaxe de l'historique d'expansion de liste de diffusion

Une valeur d'attribut mlExpansionHistory a le type ASN.1 de MLExpansionHistory. Si il y a plus de ub-ml-expansion-history listes de diffusion dans la séquence, l'agent receveur devrait donner notification de l'erreur à un administrateur humain de la liste de diffusion. L'administrateur de liste de diffusion est chargé de corriger la condition de débordement.

```
MLExpansionHistory ::= SEQUENCE  SIZE (1..ub-ml-expansion-history) OF MLData

id-aa-mlExpandHistory OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
                                              smime(16) id-aa(2) 3}

ub-ml-expansion-history INTEGER ::= 64
```

MLData contient l'historique d'expansion qui décrit chaque MLA qui a traité un message. Lorsque il distribue un message aux membres d'une liste de diffusion, le MLA enregistre son identifiant, sa date et son heure d'expansion, et la politique de réception dans une structure MLData.

```
MLData ::= SEQUENCE {
  mailListIdentifier EntityIdentifier, expansionTime GeneralizedTime,
  mlReceiptPolicy MLReceiptPolicy OPTIONAL }

EntityIdentifier ::= CHOICE { issuerAndSerialNumber IssuerAndSerialNumber, subjectKeyIdentifier SubjectKeyIdentifier }
```

La politique de réception de la liste de diffusion peut retirer la demande du générateur d'un retour de récépissé signé. Cependant, si le générateur du message n'a pas demandé de récépissé signé, le MLA ne peut pas demander un récépissé signé. Dans le cas où la politique de récépissé signé du MLA supplante la demande de récépissé signé du générateur, ce qui fait que le générateur ne va recevoir aucun récépissé signé, le MLA PEUT alors informer le générateur de ce fait.

Lorsque présent, le mlReceiptPolicy spécifie une politique de réception qui supplante la demande du générateur de récépissés signés. La politique peut être une de ces trois possibles : les récépissés NE DOIVENT PAS être retournés (aucun) ; les récépissés devraient être retournés à une autre liste de receveurs, au lieu du générateur (insteadOf) ; ou les récépissés devraient être retournés à une liste de receveurs en plus du générateur (inAdditionTo).

```
MLReceiptPolicy ::= CHOICE {
   none [0] NULL,
   insteadOf [1] SEQUENCE SIZE (1..MAX) OF GeneralNames,
   inAdditionTo [2] SEQUENCE SIZE (1..MAX) OF GeneralNames }
```

5. Signature de l'attribut Certificate

Des problèmes ont été soulevés à propos du fait que le certificat, dont le signataire d'un objet CMS SignedData désirait qu'il soit lié dans le processus de vérification de l'objet SignedData, n'est pas cryptographiquement lié dans la signature elle-même. La présente section s'adresse à cette question en créant un nouvel attribut à placer dans la section des attributs signés de l'objet SignerInfo.

Cette section présente aussi une description d'un ensemble d'attaques possibles au moyen de la substitution d'un certificat pour vérifier la signature par le certificat désiré. Un ensemble de moyens de prévenir ou de contrer ces attaques est présenté pour faire face aux plus simples des attaques.

Les informations d'autorisation peuvent être utilisées au titre du processus de vérification de la signature. Ces informations peuvent être portées dans l'un ou l'autre certificat d'attribut et d'autres certificats de clé publique. Le signataire a besoin de la capacité de restreindre l'ensemble des certificats utilisés dans le processus de vérification de signature, et les informations doivent être codées afin qu'elles soient couvertes par la signature sur l'objet SignedData. Les méthodes exposées dans cette section permettent de faire la liste de l'ensemble de certificats d'autorisation au titre de la signature de l'attribut de certificat.

Les politiques explicites de certificat peuvent aussi être utilisées au titre du processus de vérification de signature. Si un signataire désire déclarer une politique explicite de certificat qui devrait être utilisée lors de la validation de la signature, cette politique doit être liée cryptographiquement dans le processus de signature. Les méthodes décrites dans cette section permettent de faire la liste d'un ensemble de déclarations de politiques de certificat au titre de l'attribut de certificat de signature.

5.1 Descriptions d'attaques

Au moins trois attaques différentes peuvent être lancées contre un possible processus de vérification de signature en remplaçant le ou les certificats utilisés dans le processus de vérification de signature.

5.1.1 Description de l'attaque de substitution

La première attaque consiste en une simple substitution d'un certificat par un autre certificat. Dans cette attaque, le producteur et le numéro de série dans le SignerInfo sont modifiés pour se référer à un nouveau certificat. Ce nouveau certificat est utilisé durant le processus de vérification de signature.

La première version de cette attaque est une simple attaque de déni de service où un certificat invalide est substitué au certificat valide. Cela rend le message invérifiable, car la clé publique dans le certificat ne correspond plus à la clé privée utilisée pour signer le message.

La seconde version est une substitution d'un certificat valide aux certificats valides originaux lorsque les clés publiques dans les certificats correspondent. Cela permet que la signature soit validée sous des contraintes de certificat potentiellement différentes de celles que prévoyait le générateur du message.

5.1.2 Reproduction de la description du certificat

La seconde attaque est celle où une autorité de certificat (CA, *certificate authority*) reproduit le certificat de signature (ou éventuellement un de ses certificats). Cette attaque peut commencer à devenir plus fréquente car les autorités de certificat reproduisent leurs propres certificats racines, ou comme les autorités de certificat changent les politiques dans le certificat lorsque elles reproduisent leurs certificats racines. Ce problème se produit aussi lorsque des certificats croisés (avec des restrictions qui peuvent être différentes) sont utilisés dans le processus de vérification de signature.

5.1.3 Fausse description dupliquée de CA

La troisième attaque est celle d'une entité pirate qui établit une autorité de certificat qui tente de dupliquer la structure d'une CA existante. Précisément, l'entité pirate produit un nouveau certificat avec les mêmes clés publiques que celles utilisées par le

signataire, mais signées par la clé privée de l'entité pirate.

5.2 Réponses aux attaques

Le présent document ne tente pas de résoudre toutes les attaques ci-dessus ; cependant, une brève description des réponses à chaque attaque est données dans cette section.

5.2.1 Réponse à l'attaque de substitution

L'attaque de déni de service ne peut pas être empêchée. Après que l'identifiant de certificat a été modifié dans le transit, aucune vérification de la signature n'est possible. Il n'y a aucun moyen d'identifier automatiquement l'attaque parce que on ne peut pas la distinguer d'une corruption du message.

La substitution d'un certificat valide peut être contrée de deux manières différentes. La première est de faire une déclaration couverture disant que l'utilisation de la même clé publique dans deux certificats différents est une mauvaise pratique qui doit être évitée. En pratique, il n'y a pas de moyen réel d'empêcher les usagers d'obtenir de nouveaux certificats avec la même clé publique, et on devrait supposer qu'ils feront comme cela. Le paragraphe 5.4 fournit un nouvel attribut qui peut être inclus dans les attributs signés SignerInfo. Cela lie l'identifiant du certificat correct à la signature. Cela va convertir l'attaque d'un succès potentiel en une simple attaque de déni de service.

5.2.2 Réponse à la reproduction d'un certificat

Une CA ne devrait jamais reproduire un certificat avec des attributs différents. Les autorités de certificat qui le font ont une mauvaise pratique et on ne peut pas leur faire confiance. Utiliser le hachage du certificat comme référence du certificat empêche cette attaque pour les certificats d'entité d'extrémité.

Empêcher l'attaque fondée sur la reproduction des certificats de CA exigerait un changement substantiel de l'usage de l'attribut signingCertificate présenté au paragraphe 5.4. Cela exigerait que les ESSCertIDs soient inclus dans les attributs pour représenter les certificats du producteur dans le chemin de certification du signataire.Cela pose des problèmes lorsque la partie qui s'appuie sur eux utilise un certificat croisé au titre de son processus d'authentification, et ce certificat n'apparaît pas sur la liste des certificats. Les problèmes en dehors d'une PKI fermée rendent l'ajout de ces informations propices aux erreurs, causant éventuellement le rejet d'une chaîne valide.

5.2.3 Réponse à une fausse CA dupliquée

La meilleure méthode pour empêcher cette attaque est d'éviter de faire confiance à la fausse CA. L'utilisation du hachage pour identifier les certificats empêche l'utilisation de certificats d'entité d'extrémité provenant de la fausse autorité. Cependant la seule véritable façon d'empêcher cette attaque est de ne jamais faire confiance à la fausse CA.

5.3 Contexte en relation avev la vérification de signature

Certaines applications exigent que des informations supplémentaires soient utilisées au titre du processus de validation de signature. En particulier, des informations d'autorisation provenant des certificats d'attribut et d'autres certificats de clé publique ou des identifiants de politique qui fournissent des informations supplémentaires sur les capacités et les intentions du signataire. L'attribut de certificat de signature décrit au paragraphe 5.4 donne la capacité de lier ces informations de contexte au titre de la signature.

5.3.1 Informations d'autorisation

Certaines applications exigent que les informations d'autorisation trouvées dans les certificats d'attribut et/ou d'autres certificats de clé publique soient validés. Cette validation exige que l'application soit capable de trouver les certificats corrects pour effectuer le processus de vérification ; il n'y a cependant pas de liste des certificats à utiliser dans un objet SignerInfo. L'envoyeur a la capacité d'inclure un ensemble de certificats d'attribut et de certificats de clé publique dans un objet SignedData. Le receveur a la capacité de restituer les certificats d'attribut et les certificats de clé publique à partir d'un service de répertoire. Il y a des circonstances où le signataire peut souhaiter limiter l'ensemble des certificats qui peuvent être utilisés dans la vérification de signature. Il est utile d'être capable de faire la liste de l'ensemble des certificats que le signataire veut que le receveur utilise pour la validation de signature.

5.3.2 Informations de politique

Un aspect en rapport avec le lien du certificat est la question des chemins de certification multiples. Dans certaines instances, la sémantique d'un certificat dans son utilisation avec un message peut dépendre des autorités de certificat et des politiques qu'elles appliquent. Pour régler cette question, le signataire peut aussi souhaiter lier ce contexte sous la signature. Bien que ceci puisse être fait par la signature du chemin complet de certification ou par un identifiant de politique, seul un lien avec l'identifiant de politique est décrit ici.

5.4 Définition de l'attribut SigningCertificate

L'attribut de certificat de signature est conçu pour empêcher la simple attaque de substitution et de reproduction, et pour permettre d'utiliser un ensemble restreint de certificats d'autorisation dans la vérification de signature.

La définition de SigningCertificate est :

```
SigningCertificate ::=  SEQUENCE {
   certs        SEQUENCE OF ESSCertID,
   policies     SEQUENCE OF PolicyInformation OPTIONAL
}

id-aa-signingCertificate OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
                                                 smime(16) id-aa(2) 12 }
```

Le premier certificat identifié dans la séquence d'identifiants de certificats DOIT être le certificat utilisé pour vérifier la signature. Le codage de ESSCertID pour ce certificat DEVRAIT inclure le champ issuerSerial. Si d'autres contraintes assurent que issuerAndSerialNumber sera présent dans les SignerInfo, le champ issuerSerial PEUT être omis. Le certificat identifié est utilisé durant le processus de vérification de signature. Si le hachage du certificat ne correspond pas au certificat utilisé pour vérifier la signature, la signature DOIT être considérée comme invalide.

Si plus d'un certificat est présent dans la séquence de ESSCertID, les certificats après le premier limitent l'ensemble des certificats d'autorisation qui sont utilisés durant la validation de signature. Les certificats d'autorisation peuvent être des certificats d'attribut ou des certificats normaux. Le champ issuerSerial (dans la structure ESSCertID) DEVRAIT être présent pour ces certificats, sauf si le client qui valide la signature est supposé avoir un accès facile à tous les certificats exigés pour la validation. Si seul le certificat de signature est présent dans la séquence, il n'y a pas de restriction sur l'ensemble des certificats d'autorisation utilisés pour valider la signature.

La séquence des termes d'informations de politique identifie les politiques de certificat que le signataire affirme appliquer au certificat, et sous lesquelles le certificat devrait s'appuyer. Cette valeur suggère une valeur de politique à utiliser dans la validation du chemin de certification du consommateur d'assertions.

Si il est présent, l'attribut SigningCertificate DOIT être un attribut signé ; il NE DOIT PAS être un attribut non signé. La CMS définit SignedAttributes comme un SET OF Attribute *(ensemble d'attributs)*. Un SignerInfo NE DOIT PAS inclure plusieurs instances de l'attribut SigningCertificate. La CMS définit la syntaxe ASN.1 pour les attributs signés comme incluant des attrValues SET OF AttributeValue. Un attribut SigningCertificate DOIT inclure seulement une instance de AttributeValue. Il NE DOIT PAS y avoir zéro ou plusieurs instances de AttributeValue présentes dans le attrValues SET OF AttributeValue.

5.4.1 Identification du certificat

Le meilleur moyen d'identifier les certificats est une question souvent discutée. La [RFC2632] a imposé une restriction sur les objets SignedData selon laquelle le nom de domaine (DN) producteur doit être présent dans tous les certificats de signature. La paire producteur/numéro de série est donc suffisante pour identifier le certificat de signature correct. Cette information est déjà présente, au titre de l'objet SignerInfo, et la duplication de cette information serait malencontreuse. Un hachage du certificat entier assure la même fonction (permettant au receveur de vérifier qu'est utilisé le même certificat que lorsque le message a été signé) il est plus petit, et permet une détection des attaques de simple substitution.

Les certificats d'attribut et les certificats supplémentaires de clé publique contenant des informations d'autorisation n'ont pas de paire producteur/numéro de série représentée quelque part dans un objet SignerInfo. Lorsque un certificat d'attribut ou un certificat supplémentaire de clé publique n'est pas inclus dans l'objet SignedData, il devient beaucoup plus difficile d'obtenir l'ensemble correct de certificats sur la seule base du hachage du certificat. Pour cette raison, ces certificats DEVRAIENT être identifiés par l'objet IssuerSerial.

Le présent document définit un identifiant de certificat comme :

```
ESSCertID ::=  SEQUENCE {
    certHash              Hash,
    issuerSerial          IssuerSerial OPTIONAL
}

 Hash ::= OCTET STRING                          -- hachage SHA1 du certificat entier

IssuerSerial ::= SEQUENCE {
    issuer                GeneralNames,
    serialNumber          CertificateSerialNumber
}
```

Lors de la création d'un ESSCertID, le certHash est calculé sur le certificat entier codé en DER incluant la signature. Le issuerSerial va normalement être présent sauf si la valeur peut être déduite d'autres informations.

Lors du codage du IssuerSerial, serialNumber est le numéro de série qui identifie de façon univoque le certificat. Pour les certificats qui ne sont pas d'attribut, le champ issuer DOIT contenir seulement le nom du producteur à partir du certificat codé dans le choix directoryName de GeneralNames. Pour les certificats d'attribut, le champ issuer DOIT contenir le champ Nom du producteur provenant du certificat d'attribut.

6. Considérations sur la sécurité

Toutes les considérations sur la sécurité des [RFC2630] et [RFC2312] s'appliquent aux applications qui utilisent les procédures décrites dans le présent document.

Comme mentionné au paragraphe 2.3, un receveur d'une demande de récépissé ne doit pas renvoyer une réponse si il ne peut pas valider la signature. De même, si il y a des demandes de récépissé en conflit dans un message, le receveur ne doit pas renvoyer de récépissé, car un attaquant peut avoir inséré les demandes contradictoires. Envoyer un récépissé signé à un envoyeur invalidé peut exposer des informations sur le receveur qui peut ne pas vouloir les exposer à des envoyeurs inconnus.

Les envoyeurs de récépissé devraient envisager de chiffrer les récépissés pour empêcher un attaquant passif de glaner des information dans les récépissés.

Les envoyeurs ne doivent pas compter sur le logiciel de traitement des receveurs pour traiter correctement les étiquettes de sécurité. C'est-à-dire que l'envoyeur ne peut pas supposer qu'ajouter une étiquette de sécurité à un message va empêcher les receveurs de regarder les messages que l'envoyeur ne voudrait pas qu'ils regardent. Il est prévu que de nombreux clients S/MIME ne vont pas comprendre les étiquettes de sécurité mais vont quand même afficher le message étiqueté au receveur.

Un agent receveur qui traite les étiquettes de sécurité doit traiter avec soin le contenu du message. Si l'agent décide de ne pas montrer le message au receveur prévu après le traitement de l'étiquette de sécurité, l'agent doit veiller à ce que le receveur ne voit pas accidentellement le contenu ultérieurement. Par exemple, si une réponse d'erreur envoyée au générateur contient le contenu qui était caché au receveur, et si cette réponse d'erreur rebondit chez l'envoyeur à cause d'une erreur d'adressage, le receveur d'origine peut voir le contenu car il est peu probable que le message de rebond ait l'étiquette de sécurité appropriée.

Une attaque par interposition peut être cause qu'un receveur envoie des récépissés à un attaquant si celui-ci a une signature qui peut être validée par le receveur. L'attaque consiste à intercepter le message d'origine et à ajouter un attribut mLData qui dit qu'un récépissé devrait être envoyé à l'attaquant en plus de tous ceux qui devaient avoir le récépissé.

Les listes de diffusion qui chiffrent leur contenu peuvent être des cibles d'attaques de déni de service si elles n'utilisent pas la gestion de liste de diffusion décrite à la Section 4. En utilisant une simple usurpation d'en-tête de la RFC822, il est assez facile d'abonner une liste de diffusion chiffrée à une autre, établissant pas là une boucle infinie.

Les agents de liste de diffusion doivent être avertis qu'ils peuvent être utilisés comme oracles pour l'attaque de texte chiffré choisi adaptatif décrite dans la [RFC2630]. Les MLA devraient notifier à un administrateur qu'un grand nombre de messages indéchiffrables sont reçus.

Lorsque on vérifie une signature en utilisant des certificats qui viennent avec un message de la [RFC2630], le receveur devrait seulement vérifier en utilisant les certificats précédemment connus comme valides, ou les certificats qui sont venus d'un attribut signé SigningCertificate. Autrement, les attaques décrites dans la Section 5 peuvent amener le receveur à penser qu'une signature est valide alors qu'elle ne l'est pas.

A. Module ASN.1

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

DEFINITIONS DES ÉTIQUETTES IMPLICITES ::=
DÉBUT

IMPORTATIONS

-- Syntaxe de message cryptographique (CMS)
   ContentType, IssuerAndSerialNumber, SubjectKeyIdentifier FROM CryptographicMessageSyntax { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cms(1)}

-- Certificat PKIX et profil de CRL, Section A.2 Module étiqueté implicitement, Syntaxe 1988
   PolicyInformation FROM PKIX1Implicit88 {iso(1)
   identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7)id-mod(0) id-pkix1-implicit-88(2)}

-- X.509
   GeneralNames, CertificateSerialNumber FROM CertificateExtensions
   {joint-iso-ccitt ds(5) module(1) certificateExtensions(26) 0};

-- Services de sécurité étendus
-- La construction "SEQUENCE SIZE (1..MAX) OF" apparait dans plusieurs constructions ASN.1dans ce module.
-- Une SEQUENCE ASN.1 VALIDE PEUT AVOIR zéro, une ou plusieurs entrées. La construction SIZE (1..MAX) contraint
-- la SEQUENCE à avoir au moins une entrée. MAX indique que la limite supérieure est inspécifiée. Les mises en œuvre sont
-- libres de choisir une limite supérieure qui convient à leurenvironnement.

UTF8String ::= [UNIVERSAL 12] IMPLICIT OCTET STRING
-- Le contenu est formaté comme décrit dans la [RFC2279]

-- Section 2.7

ReceiptRequest ::= SEQUENCE {
  signedContentIdentifier ContentIdentifier,
  receiptsFrom ReceiptsFrom,
  receiptsTo SEQUENCE SIZE (1..ub-receiptsTo) OF GeneralNames }

ub-receiptsTo INTEGER ::= 16

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

ContentIdentifier ::= OCTET STRING

id-aa-contentIdentifier OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
                                                smime(16) id-aa(2) 7}

ReceiptsFrom ::= CHOICE {
  allOrFirstTier [0] AllOrFirstTier,                -- anciennement "allOrNone [0]AllOrNone"
  receiptList [1] SEQUENCE OF GeneralNames }

AllOrFirstTier ::= INTEGER {                        -- anciennement AllOrNone
  allReceipts (0),
  firstTierRecipients (1) }


-- Section 2.8

Receipt ::= SEQUENCE {
```

```
  version ESSVersion,
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

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

ESSVersion ::= INTEGER  { v1(1) }

-- Section 2.9

ContentHints ::= SEQUENCE {
  contentDescription UTF8String (SIZE (1..MAX)) OPTIONAL,
  contentType ContentType }

id-aa-contentHint OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
                                          id-aa(2) 4}

-- Section 2.10

MsgSigDigest ::= OCTET STRING

id-aa-msgSigDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
                                           id-aa(2) 5}

-- Section 2.11

ContentReference ::= SEQUENCE {
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

id-aa-contentReference   OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
                                                 smime(16) id-aa(2) 10 }

-- Section 3.2

ESSSecurityLabel ::= SET {
  security-policy-identifier SecurityPolicyIdentifier,
  security-classification SecurityClassification OPTIONAL,
  privacy-mark ESSPrivacyMark OPTIONAL,
  security-categories SecurityCategories OPTIONAL }

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

SecurityPolicyIdentifier ::= OBJECT IDENTIFIER

SecurityClassification ::= INTEGER {
  unmarked (0),
  unclassified (1),
  restricted (2),
  confidential (3),
  secret (4),
  top-secret (5) } (0..ub-integer-options)

ub-integer-options INTEGER ::= 256

ESSPrivacyMark ::= CHOICE {
   pString      PrintableString (SIZE (1..ub-privacy-mark-length)),
   utf8String   UTF8String (SIZE (1..MAX))
}
```

```
ub-privacy-mark-length INTEGER ::= 128

SecurityCategories ::= SET SIZE (1..ub-security-categories) OF SecurityCategory

ub-security-categories INTEGER ::= 64

SecurityCategory ::= SEQUENCE {
  type  [0] OBJECT IDENTIFIER,
  value [1] ANY DEFINED BY type                    -- défini par type
}

--Note : la syntaxe susmentionnée de SecurityCategory produit des codages hexadécimaux identiques à la syntaxe suivante de
-- SecurityCategory qui est documentée dans la spécification X.411:
-- SecurityCategory ::= SEQUENCE {
--     type  [0]  SECURITY-CATEGORY,
--     value [1]  ANY DEFINED BY type }
--
--SECURITY-CATEGORY MACRO ::=
--BEGIN
--TYPE NOTATION ::= type | empty
--VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
--END

-- Section 3.4

EquivalentLabels ::= SEQUENCE OF ESSSecurityLabel

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

-- Section 4.4

MLExpansionHistory ::= SEQUENCE
       SIZE (1..ub-ml-expansion-history) OF MLData

id-aa-mlExpandHistory OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
                                                smime(16) id-aa(2) 3}

ub-ml-expansion-history INTEGER ::= 64

MLData ::= SEQUENCE {
  mailListIdentifier EntityIdentifier,
  expansionTime GeneralizedTime,
  mlReceiptPolicy MLReceiptPolicy OPTIONAL }

EntityIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier SubjectKeyIdentifier }

MLReceiptPolicy ::= CHOICE {
  none [0] NULL,
  insteadOf [1] SEQUENCE SIZE (1..MAX) OF GeneralNames,
  inAdditionTo [2] SEQUENCE SIZE (1..MAX) OF GeneralNames }

-- Section 5.4

SigningCertificate ::=  SEQUENCE {
    certs       SEQUENCE OF ESSCertID,
    policies    SEQUENCE OF PolicyInformation OPTIONAL
}

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

```
                                    smime(16) id-aa(2) 12 }

ESSCertID ::=  SEQUENCE {
    certHash             Hash,
    issuerSerial         IssuerSerial OPTIONAL
}

Hash ::= OCTET STRING -- SHA1 hash of entire certificate

IssuerSerial ::= SEQUENCE {
    issuer               GeneralNames,
    serialNumber         CertificateSerialNumber
}

FIN                                     -- de ExtendedSecurityServices
```

B. Références

[MSP4] "Secure Data Network System (SDNS) Message Security Protocol (MSP) 4.0", Specification SDN.701, Revision A, 1997-02-06.

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

[RFC2279] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", janvier 1998. (*Obsolète, voir* RFC3629, STD 63)

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

[RFC2312] S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein, "Traitement de certificat S/MIME version 2", mars 1998. *(Info.)*

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

[RFC2630] R. Housley, "Syntaxe de message cryptographique", juin 1999. *(Obsolète, voir* RFC5652, 3370) (P.S.)

[RFC2632] B. Ramsdell, éd., "Traitement de certificat S/MIME version 3", juin 1999. (*Remplacée par* RFC5750) *(P.S.)*

[RFC2633] B. Rmasdell, "Spécification de message S/MIME version 3", juin 1999. (*Remplacée par* RFC5751) *(P.S.)*

[X.208] Recommandation UIT-T X.208, "Spécification de la notation numéro un de syntaxe abstraite (ASN.1)".

[X.411] Recommandation UIT-T X.411, "Réseaux de communication de données – Systèmes de traitement de message : systèmes de transfert de message : définition et procédures de service abstrait", 1988. MTSAbstractService {joint-iso-ccitt mhs-motis(6) mts(3) modules(0) mts-abstract-service(1)}

[X.680] Recommandation UIT-T X.680, "Spécification de la notation numéro un de syntaxe abstraite (ASN.1)".

C. Remerciements

Le premier projet de ce travail a été préparé par David Solo. John Pawling a fourni un énorme travail de révision très détaillé durant les nombreuses phases du document.

De nombreuses autres personnes ont contribué par de durs travaux au présent mémoire, et en particulier : Andrew Farrell, Bancroft Scott, Bengt Ackzell, Bill Flanigan, Blake Ramsdell, Carlisle Adams, Darren Harter, David Kemp, Denis Pinkas, Francois Rousseau, Jim Schaad, Russ Housley, Scott Hollenbeck, et Steve Dusse.

Adresse de l’éditeur

Paul Hoffman
Internet Mail Consortium
127 Segre Place
Santa Cruz, CA 95060
mél : phoffman@imc.org

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d’édition des RFC est actuellement fourni par l’Internet Society.