rfc2315

Syntaxe des messages cryptographiques

Définitions

Pour ce document, les définitions suivantes s'appliquent :

AlgorithmIdentifier un type qui identifie un algorithme et ses paramètres associés. Ce type est définis dans X.509.

ASN.1 Abstract Syntax Notation One, définis dans X.208

Attribute un type qui contient un type d'attribut et une ou plusieurs valeurs d'attribut. Ce type est définis dans X.501.

BER Basic Encoding Rules, comme définis dans X.209

Certificat un type qui lie un dn d'une entité à une clé publique avec une signature numérique. Ce type est définis dans X.509. Ce type contient également le dn du fournisseur du certificat, un numéro de série, l'algorithme de signature, et une période de validité.

CertificateSerialNumber un type qui identifie de manière unique un certificat avec ceux signés par un fournisseur de certificat particulier. Ce type est définis dans X.509.

CertificateRevocationList un type qui contient des informations sur les certificats dont la validité a été prématurément révoquée par le fournisseur.

DER Distinguished Encoding Rules for ASN.1, comme définis dans X.509

DES Data Encryption Standard, comme définis dans FIPS PUB 46-1

desCBC L'identifiant d'objet pour DES en mode cipher-block chaining

ExtendedCertificate Un type qui consiste d'un certificat X.509 et un jeu d'attributs, signés collectivement par le fournisseur du certificat. Définis dans PKCS#6

MD2 RSA Data Security, incluant l'algorithme MD2. rfc1319

md2 L'identifiant d'objet pour MD2

MD5 RSA Data Security, incluant l'algorithme MD5. rfc1321

md5 L'identifiant d'objet pour MD5

Name Un type qui identifie de manière unique des objets dans un annuaire X.500. Ce type est définis dans X.501. Dans un certificat X.509, le type identifie de fournisseur du certificat et l'entité dont la clé publique est certifiée.

PEM Internet Privacy-Enhanced, comme définis dans rfc1421-1424

RSA Le crypto-système à clé publique RSA

rsaEncryption L'identifiant d'objet pour le chiffrement RSA

Vue générale

La syntaxe est suffisamment généraliste pour supporter de nombreux types de contenus. Ce document en définis 6 : donnée, donnée signée, donnée enveloppée, donnée enveloppée et signée, donnée hashée et donnée chiffrée. D'autres types peuvent être ajoutés dans le future. L'utilisation des types de contenu définis en dehors de ce document est possible, mais est sujet à agrément bilatéral entre les parties échangeant du contenu.

Ce document exporte un type, ContentInfo, et divers identifiants d'objets.

Il y a 2 classes de type de contenu : base et amélioré. Les types de contenu dans la classe de base contient seulement les donnée, sans amélioration cryptographique. Présentement, un type de contenu est dans cette classe, le type de contenu donnée. Les types de contenus

dans la classe améliorée contiennent le contenu d'un certain type (possiblement chiffré), et d'autre améliorations cryptographique. Par exemple, le contenu donnée enveloppée peut contenir du contenu de données signée, qui peut contenir du contenu donnée. Les 4 type de contenu non données remplissent la classe améliorée. Les types de contenu dans la classe améliorée emploient ainsi l'encapsulation, donnent naissance au termes de contenu extérieur (celui contenant les améliorations) et de contenu intérieur (celui qui est amélioré).

Le document est conçu de manière à ce que les types de contenus améliorés puissent être préparés dans une simple classe en utilisant un encodage BER de longueur indéfinie. L'opération à une seule passe est spécialement utile si le contenu est stockée sur des cassettes, ou est pipé depuis un autre processus. Un des inconvénients de l'opération simple-passe est qu'il est difficile de sortir un encodé DER en une simple passe, vu que la longueur de divers composants peuvent ne pas être connus à l'avance. Vu que l'encodage DER est requis par les type de contenu donnée signée, donnée signé et enveloppée, donnée hashée, une passe supplémentaire peut être nécessaire quand un type de contenu autre que donnée est le contenu interne de l'un de ces types de contenu.

Types utiles

Les sections suivantes définissent les types qui sont utiles dans au moins 2 endroits dans le document.

CertificateRevocationLists

Le type CertificateRevocationList donne un jeu de listes de révocation de certificat. Il est conçus pour que le jeu d'informations soit suffisant pour déterminer si les certificats avec lequel le jeu est associé est "hot listed" mais il peut y avoir plusieurs listes de révocation que nécessaire, ou moins que nécessaire.

CertificateRevocationLists : := SET OF CertificateRevocationList

ContentEncryptionAlgorithmIdentifier

Le type ContentEncryptionAlgorithmIdentifier identifie un algorithme de chiffrement de contenu tel que DES. Un algorithme de chiffrement de contenu supporte les opérations de chiffrement et de déchiffrement. L'opération de chiffrement map une chaîne d'octets (le message) en une autre chaîne d'octet (le texte chiffré) sous le contrôle de la clé de chiffrement de contenu. L'opération de déchiffrement est l'inverse.

ContentEncryptionAlgorithmIdentifier : := AlgorithmIdentifier

DigestAlgorithmIdentifier

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

DigestAlgorithmIdentifier : := AlgorithmIdentifier

DigestEncryptionAlgorithmIdentifier

Le type DigestEncryptionAlgorithmIdentifier identifie un algorithme de chiffrement de hash avec lequel un hash peut être chiffré. Par

exemple, rsaEncryption. Un algorithme de chiffrement de hash supporte les opérations de chiffrement et de déchiffrement. L'opération de chiffrement mappe une chaîne d'octets (le hash) a une autre chaîne d'octets (le hash chiffré) sous le contrôle d'une clé de chiffrement de hash. L'opération de déchiffrement est l'inverse.

DigestEncryptionAlgorithmIdentifier : := AlgorithmIdentifier

ExtendedCertificateOrCertificate

Le type ExtendedCertificateOrCertificate donne soit un certificat étendu PKCS#6 ou un certificat X.509. Ce type suit la syntaxe recommandée dans PKCS#6 :

```
ExtendedCertificateOrCertificate ::= CHOICE {
 certificate Certificate, - X.509
 extendedCertificate [0] IMPLICIT ExtendedCertificate }
```

ExtendedCertificatesAndCertificates

le type ExtendedCertificatesAndCertificates donne un jeu de certificats étendus et de certificats X.509. Il est conçus pour que le jeu soit suffisant pour contenir des chaînes depuis un "root" ou "top-level certification authority" vers tous les signataires avec lequel le jeu est associé, mais il peut y avoir plus de certificats que nécessaire, ou moins.

ExtendedCertificatesAndCertificates : := SET OF ExtendedCertificateOrCertificate

IssuerAndSerialNumber

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

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

KeyEncryptionAlgorithmIdentifier

Le type KeyEncryptionAlgorithmIdentifier identifie un algorithme de chiffrement de clé avec lequel une clé de chiffrement de contenu peut être chiffrée. Ex : rsaEncryption.

KeyEncryptionAlgorithmIdentifier : := AlgorithmIdentifier

Version

Le type Version donne un numéro de version de syntaxe, pour compatibilité avec de future révisions de ce document.

Syntaxe générale

La syntaxe générale pour le contenu échangé entre les entités en accord avec ce document associe un type de contenu avec le contenu. La syntaxe devrait avoir le ContentInfo en ASN.1 :

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

ContentType ::= OBJECT IDENTIFIER
```

contentType Indique le type de contenu. C'est un OID, qui signifie que c'est un chaîne unique assignée par l'autorité qui définis le type de contenu.

content Est le contenu. Ce champ est optionnel, et si non présent, sa valeur attendu doit être fournie par d'autres moyens.

Notes

1. Les méthodes ci-dessous assument que le type de contenu peut être déterminé uniquement par le contentType, donc le type définis avec l'oid ne devrait par être un type CHOICE.
2. Quand une valeur ContentInfo est le contenu interne d'un contenu donnée signée, donnée signée et enveloppée, ou donnée hashée, un algorithme message-digest est appliqué aux octets du contenu encodé DER du champ content. Quand une valeur ContentInfo est le contenu interne d'un contenu donnée enveloppée ou donnée signée et enveloppée, un algorithme de chiffrement de contenu est appliqué aux octets d'un encodé BER à longueur définie du champ.
3. L'omission optionnelle du champ content rend possible la construction de signatures externes, par exemple, sans modification ou réplication du contenu avec lequel la signature s'applique. Dans le cas de signatures externe, le contenu à signer sera omis de la valeur ContentInfo encapsulée interne inclus dans le type de contenu donnée signée.

Type de contenu donnée

Le type de contenu de donnée est conçus pour référer à des chaînes arbitraires d'octets, tel que des fichiers texte ASCII. L'interprétation est laissée à l'application, de telles chaînes n'ont pas besoin d'une structure interne.

Data : := OCTET STRING

Type de contenu donnée signée

Le type de contenu donnée signée consiste du contenu d'un type et du hash du contenu pour 0 ou plusieurs signataires. Le hash chiffré pour un signataire est une signature numérique sur le contenu pour ce signataire. Tout type de contenu peut être signé par plusieurs signataires en parallèle. De plus, la syntaxe a un cas dégénéré dans lequel il n'y a pas de signataires sur le contenu. Le cas dégénéré fournis un moyen de diffuser des certificat et des listes de révocation de certificat.

Il est prévu que l'application typique du type de contenu donnée signée représente la signature numérique d'un signataire sur le contenu du type de contenu donnée. Une autre application typique est de diffuser des certificat et des listes de révocation de certificat.

Le processus par lequel la donnée signée est construite se fait par les étapes suivantes :

1. Pour chaque signataire, un message digest est calculé sur le contenu avec un algorithme de hashage spécifique au signataire. (Si 2 signataires emploient le même algorithme, le hash doit alors être calculé pour seulement un d'entre eux). Si le signataire authentifie

une information autre que le contenu, le hash du contenu et l'autre information sont hashé avec l'algorithme de hashage du signataire, et le résultat devient le hash.

2. Pour chaque signataire, le hash et d'autres informations associées sont chiffrées avec la clé privé du signataire.
3. Pour chaque signataire, le hash chiffré et d'autres informations spécifiques au signataire sont collectés dans une valeur SignerInfo. Les certificats et les listes de révocation de certificat pour chaque signataire, et ceux ne correspondant pas à un signataire, sont collectés dans cette étape.
4. Les algorithme de hashage pour tous les signataires et les valeur SignerInfo pour tous les signataires sont collectés ensemble avec le contenu dans une valeur SignedData.

Un destinataire vérifie les signatures en déchiffrant le hash chiffré pour chaque signataire avec la clé publique du signataire, puis compare le hash récupéré avec un hash calculé indépendamment. La clé publique du signataire est soit contenue dans un certificat inclus dans l'information du signataire, soit est référencée par un nom de fournisseur et un numéro de série spécifique au fournisseur qui identifie de manière unique le certificat pour la clé publique.

Cette section est divisée en 5 parties. La première décrit le type SignedData, la seconde décrit le type SignerInfo, la troisième et quatrième décrivent le hashage, et la dernière section est un sommaire de la compatibilité avec PEM.

type SignedData

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

```
SignedData ::= SEQUENCE {
 version Version,
 digestAlgorithms DigestAlgorithmIdentifiers,
 contentInfo ContentInfo,
 certificates
  [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
 crls
  [1] IMPLICIT CertificateRevocationLists OPTIONAL,
 signerInfos SignerInfos }

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

version Est le numéro de version de syntaxe. Devrait être à 1 pour cette version du document.

digestAlgorithms Est une collection d'identifiant d'algorithme de hashage. Il peut y avoir plusieurs élément dans la collection, incluant 0. Chaque élément identifie l'algorithme de hashage (et ses paramètres associés) sous lequel le contenu est hashé pour un signataire. La collection est conçue pour lister les algorithmes de hashage employés par tous les signataires, dans n'importe quel ordre, pour faciliter la vérification la vérification de signature en une passe.

contentInfo Est le contenu qui est signé. Il peut avoir n'importe quel type de contenu défini.

certificates Est le jeu de certificat étendu PKCS#6 et de certificats X.509. Il est prévu que le jeu soit suffisant pour contenir des chaînes depuis un root reconnus pour tous les signataires dans le champ signerInfos. Il peut y avoir plus de certificats que nécessaire, et il peut y avoir les certificats suffisants pour contenir les chaînes depuis 2 ou plusieurs autorité racine indépendants. Il peut également y avoir moins de certificats que nécessaire, s'il est prévu que la vérification des signatures aient un moyen alternatif.

clrs Est un jou de liste de révocation de certificats. Il est prévus que le jeu contienne des informations suffisantes pou déterminer si les certificats dans le champ certificates sont listés ou non, mais une telle correspondance n'est pas nécessaire. Il peut y avoir plus d'un liste de révocation que nécessaire, et il peut y en avoir moins que nécessaire.

signerInfos Est une collection d'information par signataire. Il peut y avoir plusieurs élément dans la collection, ou aucun.

notes

1. Le fait que le champ digestAlgorithms vient avant le champ contentInfo et que le champ signerInfos vient après rend possible le traitement d'une valeur SignedData en une seul passe.
2. Les différences entre la version 1 de SignedData et la version 0 (définis dans PKCS#7, version 1.4) sont les suivantes :
   - Les champs digestAlgorithms et signerInfos peuvent contenir 0 éléments dans la version 1, mais pas dans la version 0
   - Le champ crl est permis dans le version 1, mais pas dans la version 0.

Excepté pour la différence dans le numéro de version, les valeurs SignedData version 0 sont acceptable comme valeurs de version 1. Un implémentation peut cependant traiter les valeurs SignedData n'importe quelle version comme si elles étaient de la version 1.

3. Dans le cas dégénéré où il n'y a pas de signataires sur le contenu, la valeur ContentInfo à signer est sans importance. Il est recommandé dans ce cas que le type de contenu de la valeur ContentInfo à signer soit une donnée, et le champ content du ContentInfo est omis.

Type SignerInfo

Les informations par signataires sont représenté dans le type SignerInfo :

```
SignerInfo ::= SEQUENCE {
 version Version,
 issuerAndSerialNumber IssuerAndSerialNumber,
 digestAlgorithm DigestAlgorithmIdentifier,
 authenticatedAttributes
  [0] IMPLICIT Attributes OPTIONAL,
 digestEncryptionAlgorithm
  DigestEncryptionAlgorithmIdentifier,
 encryptedDigest EncryptedDigest,
 unauthenticatedAttributes
  [1] IMPLICIT Attributes OPTIONAL }

EncryptedDigest ::= OCTET STRING
```

version Est le numéro de version de syntaxe. Devrait être à 1 pour cette version du document.

issuerAndSerialNumber Spécifie le certificat du signataire (et donc de dn et la clé publique) par dn de fournisseur et numéro de série spécifique au fournisseur.

digestAlgorithm Identifie l'algorithme de hashage (et ses paramètres associés) sous lequel le contenu et les attributs authentifiés (si présent) sont hashés. Il devrait être parmis ceux dans le champ digestAlgorithms de la valeur SignerInfo supérieur.

authenticatedAttributes Est un jeu d'attributs qui sont signés (ex : authentifiés) par le signataire. Le champ est optionnel, mais il doit être présent si le type de contenu de la valeur ContentInfo à signer n'est pas une donée. Si le champ est présent, il doit contenir, au minimum, 2 attributs :

1. Un attribut de type de contenu PKCS#9 ayant comme valeur le type de contenu de la valeur ContentInfo à signer
2. Un attribut de hash PKCS#9 ayant comme valeur le hash du contenu.

D'autres type d'attributs peut être utiles ici, tel que la date de signature, également définis dans PKCS#9.

digestEncryptionAlgorithm Identifie l"algorithme de chiffrement (et ses paramètres associés) sous lequel le message digest et informations associées sont chiffrés avec la clé privée du signataire.

encryptedDigest Est le résultat du chiffrement du hash et des informations associées avec là clé privée du signataire.

unauthenticatedAttributes Est un jeu d'attributs qui ne sont pas signé par le signataire. Ce champ est optionnel. Les types d'attributs qui peuvent être utiles ici, tels que countersignatures, sont définis dans PKCS#9.

Notes

1. Il est recommandé dans l'intérêt de la compatibilité PEM que le champ authenticatedAttributes soit omis quand le type de contenu de la valeur ContentInfo est une donnée et qu'il n'y a pas d'autres attributs à signer.
2. La différence entre SignerInfo version 1 et version 0 est le processus de chiffrement du message-digest. Seul les processus compatibles PEM sont différents, reflétant les changements dans les méthodes de signature PEM. Il n'y a pas de différence dans les processus de chiffrement de message-digest non compatible PEM.

Il est suggéré que les implémentation PKCS génèrent seulement des valeurs SignedData version 1. Vu que la méthode de signature PEM avec la version 0 est obsolète.

Processus de hashage

Le processus de hashage calcule un hash sur soit le contenu à signer soit le contenu avec les attributs authentifiés du signataire. Dans les 2 cas, l'entrée initial du processus de hashage est la valeur du contenu à signer. Spécifiquement, l'entrée initiale sont les octets de contenu du champ ContentInfo encodé en DER pour lequel le processus de signature est appliqué. Seul les octets des contenus encodé DER de ce champ sont hashé, ni les octets identifiant ni les octets de longueur.

Le résultat du processus de hashage dépend de la présence ou non du champ authenticatedAttributes. Que ce champ est absent, le résultat est simplement le hash du contenu. Quand ce champ est présent, cependant, le résultat est le hash de l'encodage DER complet des valeurs d'attributs contenus dans le champ authenticatedAttributes. (le tag IMPLICIT [0] dans le champ authenticatedAttributes ne fait pas partie des valeurs d'attributs. Le tag des valeurs d'attributs est SET OF, et l'encodé DER du tag SET OF, au lieu du tag IMPLICIT [0], doit être hashé avec les octets le longueur et de contenu des valeur d'attributs). Vu que la valeur Attributes, quand le champ est présent, doit contenir comme attribut le type de contenu et le hash du contenu, ces valeurs sont indirectement inclus dans le résultat.

Quand le contenu à signer a le type de contenu donnée et que le champ authenticatedAttributes est absent, seul la valeur de la donnée est hashé. Cela a l'avantage que la longueur du contenu à signer n'a pas besoin d'être connu à l'avance dans le processus de chiffrement. Cette méthode est compatible avec PEM.

Bien que les octets identifiant et les octets de longueur ne sont pas hashés, ils restent protégés par d'autres moyens. Les octets de longueur sont protégés par la nature de l'algorithme de hashage vu qu'il est assumé infaisable de trouver 2 messages distincts ayant le même hash. De plus, en assumant que le type de contenu détermine de manière unique les octets identifiant, ces octets sont protégés implicitement dans une des 2 manières : sont par l'inclusion du type de contenu dans les attributs authentifiés, soit en utilisant l'alternative PEM qui implique que le type de contenu soit une donnée.

Note : Le fait que le hash est calculé sur la partie encodée DER ne signifie pas que DES est la méthode requise pour représenter cette partie pour le transfère des données. En effet, il est prévu que certaines implémentations de ce document peut stocker des objets autre que encodés DER, mais de telles pratiques n'affectent par le calcule du hash.

Processus de chiffrement du hash

L'entrée dans le processus de chiffrement de hash - la valeur fournie à l'algorithme de chiffrement de hash du signataire - inclus le résultat du processus de hashage et l'identifiant de l'algorithme de hashage. Le résultat du processus de chiffrement du hash est le chiffrement avec la clé privée du signataire de l'encodé DER d'une valeur de type DigestInfo :

```
DigestInfo ::= SEQUENCE {
 digestAlgorithm DigestAlgorithmIdentifier,
 digest Digest }

 Digest ::= OCTET STRING
```

digestAlgorithm Identifie l'algorithme de hash et les paramètres associés avec lequel le contenu et les attributs authentifié sont hashés. Il devrait être le même que le champ digestAlgorithm de la valeur SignerInfo supérieur.

digest est le résultat lu processus de hashage

Notes

1. La seule différence entre le processus de signature définie ici et les algorithme de signature définis dans PKCS#1 est que les signatures sont représentées en chaîne d'octets.
2. L'entrée du processus de chiffrement typiquement va avoir 30 octets ou moins. Si digestEncryptionAlgorithm est rsaEncryption de PKCS#1, cela signifie que la longueur du modulo RSA est au moins 328bits, ce qui est raisonnable et consistant avec les recommandations de sécurité.
3. Un identifiant d'algorithme de hashage est inclus dans la valeur DigestInfo pour limiter le dégâts résultant d'un algorithme de hashage compromis. Pour l'instant, supposons un adversaire qui est capable de trouver un hash MD2 donné. Cet adversaire pourrait ainsi forger une signature pour trouver un message avec le même hash, et présenter la signature précédente comme la signature sur le nouveau message. Cette attaque va réussir seulement si le signataire avec utilisé MD2, vu que la valeur DigestInfo contient l'algorithme. Si un signataire n'a jamais validé MD2 et utilise toujours MD5, alors le MD2 compromis n'affecte pas le signataire.
4. Il y a potentiellement une ambiguïté dû au fait que la valeur DigestInfo n'indique pas si le champ de hash contient seulement le hash du contenu ou le hash du champ authenticatedAttributes encodé DER complet. En d'autre termes, il est possible pour un adversaire de transformer une signature dans les attributs authentifiés à une qui apparaît être seulement dans le contenu en changeant le contenu à encoder DER du champ authenticatedAttributes, et ainsi supprimer le champ authenticatedAttributes. La transformation inverse est possible, mais nécessite que le contenu soit l'encodé DER d'une valeur d'attributs authentifiés, ce qui est peu probable. Cette ambiguïté n'est pas un nouveau problème, ni significatif, vu que le contexte va généralement empêcher les abus. En effet, il est également possible pour un adversaire de transformer une signature dans un certificat ou une liste de révocation de certificat à une qui apparaît être seulement dans le contenu donnée signée.

Compatibilité avec PEM

La compatibilité avec les type de processus MIC-ONLY et MIC-CLEAR dans PEM se produisent quand le type de contenu de ContentInfo à signer est une donnée, il n'y a par d'attributs authentifiés, l'algorithme de hash est md2 ou md5, et l'algorithme de chiffrement de hassh est rsaEncryption. Sous toutes ces conditions, le hash chiffré produit ici matche celui produit dans PEM parce que :

1. La valeur d'entrée de l'algorithme de hash dans PEM est le même que dans ce document quand il n'y a pas d'attributs authentifiés, MD2 et MD5 dans PEM sont les même que md2 et md5.
2. La valeur chiffrée avec la clé privée du signataire dans PEM est la même que dans ce document quand il n'y a pas d'attributs authentifié. RSA dans PEM est le même que rsaEncryption.

Les autres parties du type de contenu donnée signée (certificates, CRLs, algorithm identifiers, etc.) sont facilement traduisible depuis et vers leur composant PEM correspondant.

Type de contenu donnée enveloppée

Le type de contenu donnée enveloppée consiste d'un contenu chiffré de n'importe quel type et de clé de chiffrement de contenu chiffrés pour un ou plusieurs conteneurs. La combinaison du contenu chiffré et de la clé de chiffrement de contenu chiffrée pour un conteneur est une enveloppe numérique pour ce conteneur. Tout type de contenu peut être enveloppé pour autant de type de conteneurs en parallèle.

Il est prévu que l'application typique du type de contenu donnée enveloppée est de représenter une ou plusieurs enveloppe numérique de conteneurs dans le contenu de donnée, donnée hashée, ou donnée signée. Le processus par lequel une donnée enveloppée est construite est faites des étapes suivantes :

1. Une clé de chiffrement de contenu pour un algorithme de chiffrement de contenu particulier est généré aléatoirement.
2. Pour chaque conteneur, la clé de chiffrement de contenu est chiffrée avec la clé publique du conteneur
3. Pour chaque conteneur, la clé de chiffrement chiffrée et d'autres informations spécifique au conteneur sont collectées dans une valeur RecipientInfo
4. Le contenu est chiffré avec la clé de chiffrement de contenu

5. Les valeurs RecipientInfo pour tous les conteneurs sont collectés ensemble avec le contenu chiffré dans une valeur EnvelopedData

Un destinataire ouvre l'enveloppe en déchiffrant celle des clés de chiffrement de contenu chiffré avec la clé privée du destinataire et déchiffre le contenu chiffré avec la clé de chiffrement de contenu récupérée. La clé privée du destinataire est référencée par un dn de fournisseur et un numéro de série spécifique qui identifie de manière unique le certificat pour la clé publique correspondance.

Cette section est divisée en 4 parties, la première décris le type EnvelopedData, la deuxième décris le type RecipientInfo, et la troisième et quatrième décrivent les processus de chiffrement de contenu et de chiffrement de clé. Ce type de contenu n'est pas compatible avec PEM, vu que PEM implique toujours des signatures numériques, jamais des enveloppes seules.

Type EnvelopedData

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

```
EnvelopedData ::= SEQUENCE {
 version Version,
 recipientInfos RecipientInfos,
 encryptedContentInfo EncryptedContentInfo }

RecipientInfos ::= SET OF RecipientInfo

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

EncryptedContent ::= OCTET STRING
```

version Est le numéro de version de syntaxe. Il devrait être 0 pour cette version du document

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

encryptedContentInfo Est l'information de contenu chiffré

contentType Indique le type de contenu

contentEncryptionAlgorithm Identifie l'algorithme de chiffrement de contenu et ses paramètres associés sous lequel le contenu est chiffré. Le processus de chiffrement de contenu est décris plus bas. Cet algorithme est le même pour tous les destinataires.

encryptedContent Est le résultat du chiffrement du contenu. Ce champ est optionnel, et si non présent, sa valeur doit être fournie par d'autres moyens.

Note : Le fait que le champ recipientInfos vient avant le champ encryptedContentInfo rend possible le traitement d'une valeur EnvelopedData en une seule passe.

Type RecipientInfo

Les informations par destinataire sont représentés dans le type RecipientInfo :

```
RecipientInfo ::= SEQUENCE {
 version Version,
 issuerAndSerialNumber IssuerAndSerialNumber,
 keyEncryptionAlgorithm

  KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }
```

```
EncryptedKey ::= OCTET STRING
```

version Est le numéro de version de syntaxe. Il devrait être 0 pour cette version du document

issuerAndSerialNumber Spécifie le certificat du destinataire (et donc sont dn et sa clé publique) par dn de fournisseur et numéro de série.

keyEncryptionAlgorithm Identifie l'algorithme de chiffrement de clé (et ses paramètres associés) avec lequel la clé de chiffrement de contenu est chiffrée avec la clé publique du destinataire.

encryptedKey Est le résultat du chiffrement de la clé de chiffrement de contenu avec la clé publique du destinataire.

Processus de chiffrement de contenu

L'entrée pour le processus de chiffrement de contenu est la valeur du contenu à envelopper. Spécifiquement, l'entrée sont les octets du contenu de longueur définie encodée BER du champ content de la valeur ContentInfo pour lequel le processus d'enveloppe est appliqué. Seul les octets de contenu de l'encodé BER sont chiffrés, et non les octets identifiant ou les octets de longueur ; ces octets ne sont pas représentés du tout.

Quand le contenu à envelopper à un contenu de type donnée, seule la valeur de la donnée (par ex. le contenu d'un fichier) est chiffré. Cela a l'avantage que la longueur du contenu à chiffrer n'a pas besoin d'être connus à l'avance. Cette méthode est compatible avec PEM.

Les octets identifiant et les octets de longueur ne sont pas chiffrés. Les octets de longueur peuvent être protégés implicitement par le processus de chiffrement, dépendant de l'algorithme de chiffrement. Les octets identifiant ne sont pas protégés du tout, bien qu'ils puissent être récupérés depuis le type de contenu, assumant que le type de contenu détermine de manière unique les octets identifiant. La protection explicite de l'identifiant et de la longueur nécessite que le type donnée signée et enveloppée soit employé, ou que les type de contenu donnée hashé, donnée enveloppée soient appliqués successivement.

Notes

1. La raison qu'un encodé BER de longueur définie est requise est que le bit indiquant si la longueur est définie ou indéfinie n'est pas enregistrée dans le type de contenu donnée enveloppée. L'encodage de longueur définie est plus appropriée pour des types simple tels que les chaîne d'octets.
2. Certains algorithmes de chiffrement de contenu assument que la longueur d'entrée est un multiple de k octets, où k > 1, et laissent l'application définir une méthode pour manipuler l'entrée dont la longueur ne serait pas un multiple de k octets. Pour de tels algorithmes, la méthode devrait être de compléter l'entrée à la fin avec k - (l mod k) octets, tous ayant la valeur k - (l mod k), où l est la longueur de l'entrée. En d'autre termes, l'entrée est remplie à la fin avec une des chaînes suivante :

```
01 - if l mod k = k-1
02 02 - if l mod k = k-2

 k k ... k k - if l mod k = 0
```

Le padding peut être supprimé de manière non-ambigu vu que toute l'entrée est remplie et qu'aucune chaîne de padding n'est un suffixe d'un autre. Cette méthode est définie si et seulement si k < 256 ; les méthodes pour k supérieur sont un problème.

Processus de chiffrement de clé

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

Type de contenu donnée signée et enveloppée

Cette section définis le type de contenu donnée signée et enveloppée. Pour faire court, le principal de cette section sont exprimée en terme des 2 sections précédentes.

Le type de contenu donnée signée et enveloppée consiste d'un contenu chiffré de n'importe quel type, des clés de chiffrement de contenu chiffré pour un ou plusieurs destinataires, et du hash doublement chiffrés pour un ou plusieurs signataires. Le double chiffrement consiste d'un chiffrement avec la clé privée du signataire suivi par un chiffrement avec la clé de chiffrement de contenu.

La combinaison du contenu chiffré et de la clé de chiffrement de contenu chiffré pour un destinataire est une enveloppe numérique pour ce destinataire. Le hash chiffré récupéré pour un signataire est une signature numérique sur le contenu récupéré pour ce signataire. Tout tupe de contenu peut être enveloppé pour des destinataire et signé par des signataire en parallèle.

Il est prévu que l'application typique d'un type de contenu donnée enveloppée et signée est de représenter la signature numérique d'un signataire et une ou plusieurs enveloppe numérique d'un ou plusieurs destinataires dans le contenu du type de contenu donnée. Le processus pour signer et envelopper est construit selon les étapes suivantes :

1. Une clé de chiffrement pour un algorithme de chiffrement de contenu particulier est généré aléatoirement.
2. Pour chaque destinataire, la clé de chiffrement de contenu est chiffrée avec la clé publique du destinataire
3. Pour chaque destinataire, la clé de chiffrement de contenu chiffrée et d'autres informations spécifiques au destinataire sont collectés dans une valeur RecipientInfo.
4. Pour chaque signataire, un hash est calculé sur le contenu avec un algorithme de hash spécifique au signataire. (si 2 signataires emploient le même algorithme de hash, le hash doit être calculé une seule fois).
5. Pour chaque signataire, le hash est les informations associées sont chiffrées avec la clé privée du signataire, et le résultat est chiffré avec la clé de chiffrement de contenu. (le deuxième chiffrement peut nécessiter que le résultat du premier chiffrement soit padded à un multiple d'un taille de block)
6. Pour chaque signataire, le hash et les informations spécifique doublement chiffrés sont collecté dans une valeur SignerInfo.
7. Le contenu est chiffré avec la clé de chiffrement de contenu.
8. Les algorithmes de hash pour tous les signataires, les valeurs SignerInfo pour tous les signataires et les valeurs RecipientInfo pour tous les destinataires sont collecté ensemble avec le contenu chiffré dans une valeur SignedAndEnvelopedData.

Un destinataire ouvre les enveloppes et vérifie les signature en 2 étapes. D'abord, celle contenant les clé de chiffrement de contenu est déchiffrée avec la clé privée du destinataire, et le contenu chiffré est déchiffré avec la clé de chiffrement de contenu récupéré. Ensuite, le hash doublement chiffré pour chaque signataire est déchiffré avec la clé de chiffrement de contenu récupérée, le résultat est déchiffré avec la clé publique du signataire et le hash récupéré est comparé avec un hash calculé indépendemment.

Cette section est divisée en 3 parties. La première décris le type SignedAndEnvelopedData et la deuxième décris le processus de chiffrement. Les autres types et processus sont les même que dans les sections précédente. La troisième partie est un sommaire de compatibilité avec PEM.

Note : Le type de contenu donnée signée et enveloppée fournis un amélioration cryptographique similaire à celui résultant d'un combinaison séquentielle des type de contenu donnée signée et donnée enveloppée. Cependant, vu que le type de contenu donnée signée et enveloppée n'a pas d'attributs authentifié ou non-authentifié, ni ne fournis d'enveloppe pour les information de signataire autre que la signature, la combinaison séquentielle est généralement préférable au contenu SignedAndEnvelopedData. Excepté quand la compatibilité avec le type de processus ENCRYPTED dans PEM est recherché.

Type SignedAndEnvelopedData

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

```
SignedAndEnvelopedData ::= SEQUENCE {
 version Version,
 recipientInfos RecipientInfos,
 digestAlgorithms DigestAlgorithmIdentifiers,
```

```
encryptedContentInfo EncryptedContentInfo,
certificates
 [0] IMPLICIT ExtendedCertificatesAndCertificates
  OPTIONAL,
crls
 [1] IMPLICIT CertificateRevocationLists OPTIONAL,
signerInfos SignerInfos }
```

version Est le numéro de version de syntaxe. Il devrait être 0 pour cette version du document

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

digestAlgorithms Est une collection d'identifiants d'algorithme de hashage.

encryptedContentInfo Est le contenu chiffré. Il peut avoir tout type de contenu

certificates Est un jeu de certificats PKCS#6 et de certificats X.509

crls Est un jeu de listes de révocation de certificat

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

Notes

1. Le fait que les champs recipientInfos et digestAlgorithms soient avant les champs contentInfo et signerInfos rend possible de traiter un valeur SignedAndEnvelopedData en une passe.
2. La différence entre les version 0 et 1 de SignedAndEnvelopedData est que le champ crls est permis dans la version 1, mais pas dans la version 0.

Processus de chiffrement de hash

L'entrée du processus de chiffrement est la même que pour processus de chiffrement de hash, mais le processus lui-même est différent. Spécifiquement, le processus à 2 étapes. D'abord, l'entrée est fournie à l'algorithme de chiffrement de hash du signataire. Ensuite, le résultat est chiffré avec la clé de chiffrement de contenu. Il n'y a pas d'encodage DER entre les 2 étapes ; la valeur de sortie de la première étape est entrée directement dans la seconde étape. Ce processus est compatible avec le processus ENCRYPTED dans PEM.

Note : Le but de la seconde étape est d'empêcher un adversaire de récupérer le hash du contenu. Sinon, un adversaire sera capable de déterminer qui d'une liste de contenus candidats est le contenu actuel en comparant leur hash.

Compatibilité avec PEM

La compatibilité avec le type de processus ENCRYPTED de PEM se produit quand le type de contenu de la valeur ContentInfo à signer et envelopper est une donnée, l'algorithme de hash est md2 ou md5, l'algorithme de chiffrement est DES en mode CBC, l'algorithme de chiffrement de hash est rsaEncryption, et l'algorithme de chiffrement de clé est rsaEncryption. Sous toutes ces conditions, le hash doublement chiffré et la clé de chiffrement de contenu chiffré matchent celui produit dans PEM parce que les raisons données plus haut sont similaires aussi bien que les suivantes :

1. La valeur d'entrée de l'algorithme de chiffrement de contenu dans PEM est la même que dans ce document. DES en mode CBC est le même que desCBC.
2. La valeur d'entrée de l'algorithme de chiffrement de clé dans PEM est la même que dans ce document. RSA dans PEM est le même que rsaEncryption.
3. Le processus de double chiffrement appliqué au hash dans ce document et dans PEM sont les même

Les autres parties du type de contenu donnée signée et enveloppée (certificates, CRLs, algorithm identifiers, etc.) sont facilement transposables depuis et vers leur composant PEM.

Type de contenu donnée hashé

Le type de contenu donnée hashée consiste du contenu de n'importe quel type et un hash du contenu. Il est prévu que l'application typique du type de contenu donnée hashée est d'ajouter l'intégrité du contenu d'entrée au type de contenu donnée enveloppée. Le processus par lequel une donnée hashée est construite est constituée des étapes suivantes :

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

Un destinataire vérifie le hash en comparant ce hash avec un hash calculé indépendamment.

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

```
DigestedData ::= SEQUENCE {
 version Version,
 digestAlgorithm DigestAlgorithmIdentifier,
 contentInfo ContentInfo,
 digest Digest }

Digest ::= OCTET STRING
```

version Est le numéro de version de syntaxe. Il devrait être 0 pour cette version du document

digestAlgorithm Identifie l'algorithme de hashage et ses paramètres associés avec lequel le contenu est hashé.

contentInfo Est le contenu qui est hashé

digest Est le résulta du processus de hashage

Note : Le fait que le champ digestAlgorithm vient avant le champ contentInfo est le champ digest vient aprés rend possible le traitement d'une valeur DigestedData en une seule passe.

Type de contenu donnée chiffrée

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

Il est prévu que l'application typique du type de contenu donnée chiffrée est de chiffrer le contenu du type de contenu donnée pour stochage local, où la clé de chiffrement peut être un mot de passe.

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

```
EncryptedData ::= SEQUENCE {
 version Version,
 encryptedContentInfo EncryptedContentInfo }
```

version Est le numéro de version de syntaxe. Il devrait être 0 pour cette version du document

encryptedContentInfo contient les information sur le contenu chiffré

Object Identifiers

Ce document définis 7 identifiant d'objet : pkcs-7, data, signedData, envelopedData, signedAndEnvelopedData, digestedData, et encryptedData.

L'identifiant d'objet pkcs-7 identifie ce document

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

Les OID data, signedData, envelopedData, signedAndEnvelopedData, digestedData, et encryptedData, identifient, respectivement, les type de contenu donnée, donnée signée, donnée enveloppée, donnée signée et enveloppée, donnée hashée et donnée chiffrée.

```
data OBJECT IDENTIFIER ::= { pkcs-7 1 }
 signedData OBJECT IDENTIFIER ::= { pkcs-7 2 }
 envelopedData OBJECT IDENTIFIER ::= { pkcs-7 3 }
 signedAndEnvelopedData OBJECT IDENTIFIER ::= { pkcs-7 4 }
 digestedData OBJECT IDENTIFIER ::= { pkcs-7 5 }
 encryptedData OBJECT IDENTIFIER ::= { pkcs-7 6 }
```

Ces OID sont prévus pour être utilisé dans le champ contentType d'une valeur de type ContentInfo.