rfc3370

Algorithmes de Syntaxe de message cryptographique

Introduction

CMS est utilisé pour signer, hasher, authentifier, ou chiffrer numériquement des messages. Ce document décris l'utilisation des algorithmes utilisés communément avec CMS. Les implémentations de CMS peuvent supporter ces algorithmes ; et peuvent également supporter d'autres algorithmes. Cependant, si une implémentation choisis de supporter un des algorithmes décris dans ce document, l'implémentation doit le faire comme décris dans ce document.

Les valeurs CMS sont générées en utilisant ASN.1, encodé BER. Les identifiants d'algorithmes (qui incluent les identifiants d'objets ASN.1) identifient les algorithmes de cryptographiques, et certains algorithmes nécessitent des paramètres additionnels. Quand nécessaire, les paramètres sont spécifiés avec une structure ASN.1. L'identifiant d'algorithme pour chaque algorithme est spécifié, et que nécessaire, la structure de paramètres est spécifiée. Les champs dans le CMS employé par chaque algorithme sont identifié.

Changements depuis la rfc 2630

Ce document obsolète la section 12 de la rfc 2630. La séparation des spécification du protocole et des algorithmes permet à chacun d'être mis à jours sans impacter d'autre. Cependant, les conventions pour utiliser des algorithmes additionnels avec CMS sont spécifiés dans des documents séparés.

Algorithmes de hashage

Cette section spécifie les conventions employées par les implémentations CMS qui supportent SHA-1 ou MD5. Les identifiants d'algorithmes de hashage sont localisés dans le champ digestAlgorithms du SignedData, le champ digestAlgorithm du SignerInfo, le champ digestAlgorithm du DigestedData, et le champ digestAlgorithm du AuthenticatedData.

Les valeurs de hash sont placés dans le champ DigestedData et l'attribut authentifié messages-digest.

SHA-1

L'algorithme de hashage SHA-1 est définis dans FIPS pub 180-1. L'identifiant d'algorithme pour SHA-1 est :

```
sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 26 }
```

Il y a 2 encodages possible pour le champ de paramètres AlgorithmIdentifier. Les 2 alternatives surviennent du fait que quand la syntaxe 1988 pour AlgorithmIdentifier a été traduite en syntaxe 1997, l'optionnel associé avec les paramètres AlgortihmIdentifier ont été perdus. Plus tard, cet optionnel a été récupérée, mais jusque là beaucoup de gens pensaient que les paramètres de l'algorithme étaient obligatoires. À cause de cela certaines implémentations encodent les paramètres en éléments NUL et d'autres les omettent entièrement. L'encodage correct est d'omettre le champ parameters ; cependant, les implémentations doivent également gérer les champ parameters qui contiennent un NULL.

Le champ parameters de AlgorithmIdentifier est optionnel. Si présent, le champ parameters doit contenir un NULL. Les implémentations doivent accepter un AlgorithmIdentifiers SHA-1 sans paramètres. Les implémentations doivent accepter AlgorithmIdentifiers SHA-1 avec un AlgorithmIdentifiers avec un parameters NULL. Les implémentations devraient générer des AlgorithmIdentifiers SHA-1 sans parameters.

MD5

L'algorithme de hashage MD5 est définis dans la rfc1321. L'identifiant d'algorithme pou MD5 est :

```
md5 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) digestAlgorithm(2) 5 }
```

Le champ parameters de AlgorithmIdentifier doit être présent, et doit contenir NULL. Les implémentations peuvent accepter le AlgorithmIdentifiers MD5 avec parameters absent ou à NULL.

Algorithmes de signature

Cette section spécifie les conventions employées par les implémentation CMS qui supportent DSA ou RSA. Les identifiants d'algorithme de signature sont localisés dans le champ signatureAlgorithm du SignerInfo du SignedData. Également, les identifiants d'algorithme de signature sont localisés dans le champ signatureAlgorithm du SignerInfo des attributs countersignatures. Les valeurs de signature sont localisées dans le champ signature du SignerInfo du SignedData. Également, les valeurs de signature dans le champ signature des attributs countersignature.

DSA

L'algorithme de signature DSA est définis dans FIPS Pub 186. DSA doit être utilisé avec l'algorithme SHA-1. L'identifiant d'algorithme DSA sujetà à clé publique dans les certificats est :

```
id-dsa OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) x9-57 (10040) x9cm(4) 1 }
```

La validation de signature DSA nécessite 3 paramètres, communément appelés p, q et g. Quand l'identifiant d'algorithme id-dsa est utilisé, le champ parameters de AlgorithmIdentifier est optionnel. Si présent, ce champ doit contenir les 2 valeurs de paramètre DSA encodés en utilisant le type Dss-Parms. Si absent, le sujet de la clé publique DSA utilise les même paramètres DSA que le fournisseur de certificat :

```
Dss-Parms ::= SEQUENCE {
 p INTEGER,
 q INTEGER,
 g INTEGER }
```

Quand l'identifiant d'algorithme id-dsa est utilisé, la clé publique DSA, communément appelée Y, doit être encodée en integer. La sortie de cet encodage est placé dans la clé publique du sujet du certificat :

```
Dss-Pub-Key ::= INTEGER - Y
```

L'identifiant de l'algorithme DSA avec signature SHA-1 est :

```
id-dsa-with-sha1 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) x9-57 (10040) x9cm(4) 3 }
```

Quand l'identifiant de l'algorithme id-dsa-with-sha1 est utilisé, le champ de paramètres AlgorithmIdentifier doit être absent. En signant, l'algorithme DSA génère 2 valeurs, communément appelés r et s. Pour transférer ces 2 valeurs en tant que signature, il doivent être encodés

en utilisant le type Dss-Sig-Value :

```
Dss-Sig-Value ::= SEQUENCE {
 r INTEGER,
 s INTEGER }
```

RSA

L'algorithme de signature RSA est définis dans la rfc2437, et spécifie l'utilisation de l'algorithme de signature RSA avec les algorithmes de hashage SHA-1 et MD5. L'identifiant d'algorithme pour les sujets de clé publique RSA dans les certificats est :

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

Quand l'identifiant d'algorithme rsaEncryption est utilisé, le champ parameters de AlgorithmIdentifier doit contenir NULL. Quand l'identifiant d'algorithme rsaEncryption est utilisé, la clé publique RSA, qui est composée d'un modulo et d'un exposant publique, doit être encodé en utilisant le type RSAPublicKey. La sortie de cet encodage est placé dans la clé publique du sujet du certificat.

```
RSAPublicKey ::= SEQUENCE {
 modulus INTEGER, - n
 publicExponent INTEGER } - e
```

Les implémentations CMS qui incluent l'algorithme de signature RSA doivent implémenter l'algorithme de hashage SHA-1. De telles implémentations devraient également supporter l'agorithme MD5.

L'identifiant d'algorithme rsaEncryption est utilisé pour identifier les valeurs de signature RSA sans regarder l'algorithme de hashage utilisé. Les implémentations CMS qui incluent l'algorithme de signature RSA doivent supporter l'identifiant d'algorithme de valeur de signature rsaEncryption, et les implémentations CMS peuvent supporter les identifiant d'algorithme de valeur de signature RSA et l'algorithme de hashage.

L'identifiant d'algorithme pour RSA avec signature SHA-1 est :

```
sha1WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
5 }
```

L'identifiant d'algorithme pour RSA avec signature MD5 est :

```
md5WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 4
}
```

Quand les identifiants d'algorithme de signature rsaEncryption, sha1WithRSAEncryption, ou md5WithRSAEncryption sont utilisés, le champ parameters de AlgorithmIdentifier doit être NULL.

En signant, l'algorithme RSA génère une seule valeur, et cette valeur est utilisée directement comme valeur de signature.

Algorithmes de gestion de clé

CMS accueille les techniques de gestion de clé suivant : key agreement, key transport, previously distributed symmetric key-encryption keys, et passwords.

Cette section spécifie les conventions employées par les implémentations CMS qui supportent l'agrément de clé en utilisant X9.42 Ephemeral-Static Diffie-Hellman (X9.42 E-S D-H) et X9.42 Static-Static Diffie-Hellman (X9.42 S-S D-H).

Quand un algorithme d'agrément de clé est utilisé, un algorithme de chiffrement de clé est également nécessaire. Donc, quand l'agrément

de clé est supporté, un algorithme de chiffrement de clé doit être fournis pour chaque algorithme de chiffrement de contenu. Les algorithmes d'enveloppement de clé pour Triple-DES et RC2 sont décris dans la rfc3217.

Pour l'agrément de clé des clés de chiffrement de clé RC2, 128bits doivent être générés en entrée du processus d'expansion de clé utilisé pour calculé la clé effective RC2.

Les identifiants d'algorithme d'agrément de clé sont localisé dans les champs EnvelopedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm et AuthenticatedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm.

Les identifiants d'algorithme d'envelopement de clé sont localisés dans parameters de KeyWrapAlgorithm dans les champs EnvelopedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm et AuthenticatedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm.

Les clés de chiffrement de contenu enveloppés sont localisés dans le champ EnvelopedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey. Les clés message-authentication enveloppés sont localisés dans le champ AuthenticatedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey.

X9.42 Ephemeral-Static Diffie-Hellman

L'agrément de clé Ephemeral-Static Diffie-Hellman est définis dans la rfc 2631. En utilisant Ephemeral-Static Diffie-Hellman, le champ EnvelopedData RecipientInfos KeyAgreeRecipientInfo est utilisé comme suit :

- la version doit être 3
- originator doit être l'alternative originatorKey. le champ algorithm de originatorKey doit contenir l'identifiant d'objet dh-public-number avec parameters absent. le champ publicKey de originatorKey doit contenir la clé publique éphémère de l'émetteur. l'identifiant d'objet dh-public-number est :

```
dh-public-number OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-x942(10046) number-type(2) 1 }
```

- ukm peut être présent ou absent. Les implémentations CMS doivent supporter l'absence de ukm, et devraient supporter la présence d'ukm. Quand présent, ukm est utilisé pour s'assurer qu'une clé de chiffrement de clé est générée quand la clé privée éphémère doit être utilisée plus d'une fois.
- keyEncryptionAlgorithm doit être l'identifiant d'algorithme id-alg-ESDH. Le champ de paramètre d'identifiant d'algorithme pour id-alg-ESDH est KeyWrapAlgorithm, et ce paramètre doit être présent. KeyWrapAlgorithm dénote l'algorithme de chiffrement symétrique utilisé pour chiffrer la clé de chiffrement de contenu ave la clé de chiffrement générée en utilisa l'algorithme d'agrément de clé X9.42 Ephemeral-Static Diffie-Hellman. Triple-DES et RC2 sont décris dans la rfc3217. la syntaxe de l'identifiant d'algorithme id-alg-ESDH et ses paramètres est :

```
id-alg-ESDH OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
alg(3) 5 }

KeyWrapAlgorithm ::= AlgorithmIdentifier
```

- recipientEncryptedKeys contient un identifiant et un clé chiffrée pour chaque destinataire. Le KeyAgreeRecipientIdentifier de RecipientEncryptedKey doit contenir soit le issuerAndSerialNumber identifiant le certificat du destinataire, soit RecipientKeyIdentifier contenant d'identifiant de clé du sujet du certificat du destinataire. Dans les 2 cas, le certificat du destinataire contient la clé publique statique du destinataire. EncryptedKey de RecipientEncryptedKey doit contenir la clé de chiffrement de contenu chiffrée avec la clé de chiffrement de clé X9.42 Ephemeral-Static Diffie-Hellman générée en utilisant l'algorithme spécifié par le KeyWrapAlgorithm.

X9.42 Static-Static Diffie-Hellman

L'agrément de clé Static-Static Diffie-Hellman est définis dans la rfc 2631. En utilisant Static-Static Diffie-Hellman, les champs EnvelopedData RecipientInfos KeyAgreeRecipientInfo et AuthenticatedData RecipientInfos KeyAgreeRecipientInfo sont utilisé comme suit :

- la version doit être 3
- originator doit être sost issuerAndSerialNumber soit subjectKeyIdentifier. Dans les 2 cas, le certificat de l'auteur contient la clé publique statique de l'émetteur. La rfc 3279 spécifie la syntaxe et les valeurs des paramètres AlgorithmIdentifier qui sont inclus dans le certificat de l'auteur. Le champ d'information de clé publique du certificat de l'auteur doit contenir l'identifiant d'objet dh-public-number :

```
dh-public-number OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-x942(10046) number-type(2) 1 }
```


```
dh-public-number OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-x942(10046) number-type(2) 1 }
```


- ukm doit être présent. ukm s'assure qu'une clé de chiffrement de clé différente est générée pour chaque message entre les même émetteur et destinataire.
- keyEncryptionAlgorithm doit être d'identifiant d'algorithme id-alg-SSDH. Le champ parameter de d'identifiant d'algorithme pour id-alg-SSDH est KeyWrapAlgorithm, et ce paramètre doit être présent. KeyWrapAlgorithm dénote l'algorithme de chiffrement symétrique utilisé pour chiffrer la clé de chiffrement de contenu avec la clé de chiffrement de clé générée en utilisant l'algorithme d'agrément de clé X9.42 Static-Static Diffie-Hellman. Triple-DES et RC2 sont décris dans la rfc 3217. L'identifiant d'algorithme id-alg-SSDH et ses paramètres est :

```
id-alg-SSDH OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
alg(3) 10 }

KeyWrapAlgorithm ::= AlgorithmIdentifier
```

- recipientEncryptedKeys contient un identifiant et une clé chiffrée pour chaque destinataire. KeyAgreeRecipientIdentifier de RecipientEncryptedKey doit contenir soit le issuerAndSerialNumber identifiant le certificat du destinataire, soit le RecipientKeyIdentifier contenant l'identifiant de clé du sujet du certificat du destinataire. Dans les 2 cas, le certificat du destinataire contient la clé publique statique du destinataire. EncryptedKey de RecipientEncryptedKey doit contenir la clé de chiffrement de contenu chiffrée avec la clé de chiffrement de clé X9.42 Static-Static Diffie-Hellman générée en utilisant l'algorithme spécifiée par KeyWrapAlgorithm.

Algorithmes de transport de clé

Cette section spécifie les conventions employés par les implémentations CMS qui supportent le transport de clé en utilisant RSA. Les identifiant d'algorithme de transport de clé sont localisé dans le champ EnvelopedData RecipientInfos KeyTransRecipientInfo keyEncryptionAlgorithm. Les clé de chiffrement de contenu chiffrés par transport de clé sont localisés dans le champ EnvelopedData RecipientInfos KeyTransRecipientInfo encryptedKey.

RSA

L'algorithme de transport de clé RSA est le shéma de chiffrement RSA définis dans la rfc 2313, le type de block est 02, où le message à chiffrer est la clé de chiffrement de contenu. L'identifiant d'algorithme pour RSA est :

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

Le champ parameters de AlgorithmIdentifier doit être présent, et le champ parameters doit contenir NULL. En utilisant une clé de chiffrement de contenu Triple-DES, les implémentations CMS doivent ajuster les bits de parité pour chaque clé DES comprises dans la clé Triple-DES avant le chiffrement RSA.

L'utilisation du chiffrement RSA, comme définis dans la rfc 2313, pour fournir la confidentialité a une vulnérabilité conue. La vulnérabilité est plus dans l'utilisation dans les application intéractives que dans les environnement de stockages. Dans informations et contre-mesure sont discutés dans la section considération de sécurité de ce document et la rfc3218.

Noter que le même schéma de chiffrement RSA est également définis dans la rfc 2437, qui est appelé RSAES-PKCS1-v1.5.

Algorithmes de clé de chiffrement de clé symétriques

Cette section spécifie les conventions employés par les implémentations CMS qui supportent la gestion de clé de chiffrement de clé en utilisant Triple-DES ou RC2. Quand RC2 est supporté, des clé RC2 128 bits doivent être utilisé, et doivent être utilisées avec le paramètre RC2ParameterVersion à 58. Une implémentation CMS peut supporter un key-encryption et content-encryptionalgorithms mixés. Par exemple, une clé de chiffrement de contenu 40-bits RC2 peut être enveloppé avec une clé de chiffrement de clé Triple-DES 168-bits ou avec une clé de chiffrement de clé 128-bits RC2.

Les identifiants d'algorithme d'enveloppement sont localisé dans les champs EnvelopedData RecipientInfos KEKRecipientInfo keyEncryptionAlgorithm et AuthenticatedData RecipientInfos KEKRecipientInfo keyEncryptionAlgorithm.

Les clé de chiffrement de contenus enveloppés sont localisés dans le champ EnvelopedData RecipientInfos KEKRecipientInfo. Les clé d'authentification de message enveloppés sont localisés dans le champ AuthenticatedData RecipientInfos KEKRecipientInfo encryptedKey.

La sortie d'un algorithme d'agrément de clé est une clé de chiffrement de clé, et cette clé de chiffrement de clé est utilisée pour chiffrer la clé de chiffrement de contenu. Pour suppporter l'agrément de clé, les identifiants d'algorithme d'enveloppement sont localisés dans le KeyWrapAlgorithm parameter des champs EnvelopedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm et AuthenticatedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm. Cependant, seuls les algorithmes d'agrément de clé qui fournissent des clés d'authentification doivent être utilisé avec AuthenticatedData. Les clés de chiffrement de contenu enveloppés sont localisés dans le champ EnvelopedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey, les clé d'authentification de message enveloppés sont localisés dans le champ AuthenticatedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey.

Triple-DES Key Wrap

Une implémentation CMS peut supporter des algorithmes de chiffrement de clé et de chiffrement de contenu mixés. Par exemple, une clé de chiffrement de contenu 128-bits RC2 peut être enveloppé avec une clé de chiffrement de clé 168-bits Triple-DES.

Le chiffrement Triple-DES a l'identifiant d'algorithme :

```
id-alg-CMS3DESwrap OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) alg(3) 6 }
```

Le champ parameter de AlgorithmIdentifier doit être NULL. L'algorithme d'envelope de clé utilisé pour chiffrer une clé de chiffrement de contenu Triple-DES avec une clé de chiffrement de clé Triple-DES est spécifiée dans la rfc3217.

RC2 Key Wrap

Une implémentation CMS peut supporter des algorithmes de chiffrement de clé et de chiffrement de contenu mixés. Par exemple, une clé de chiffrement de contenu 128-bits RC2 peut être enveloppé dans une clé de chiffrement de clé Triple-DES 168-bits. Similairement, une clé de chiffrement de contenu RC2 40-bits peut être enveloppé dans une clé de chiffrement de clé RC2 128-bits.

Le chiffrement de clé RC2 a l'identifiant d'algorithme :

```
id-alg-CMSRC2wrap OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) alg(3) 7 }
```

Le champ parameter de AlgorithmIdentifier doit être RC2wrapParameter :

```
RC2wrapParameter ::= RC2ParameterVersion
RC2ParameterVersion ::= INTEGER
```

Les bits de clé effectifs RC2 (la taille de clé) supérieurs à 32 et inférieurs à 256 sont encodés dans le RC2ParameterVersion. Pour les tailles 40, 64, et 128, les valeurs RC2ParameterVersion sont 160, 120, et 58, respectivement. Ces valeurs ne sont pas simplement la longueur de clé RC2. Noter que la valeur 160 doit être encodé en 2 octets (00 A0), parce que l'octet A0 seul représente un nombre négatif.

Les clé RC2 128-bits doivent être utilisés comme clé de chiffrement de clé, et doivent être utilisé avec le paramètre RC2ParameterVersion à 58.

L'algorithme d'enveloppe de clé utilisé pour chiffrer une clé de chiffrement de contenu RC2 avec une clé de chiffrement de clé RC2 est spécifiée dans la rfc 3217.

Algorithmes de dérivation de clé

Cette section spécifie les conventions employées par les implémentations CMS qui supportent la gestion de clé basé sur mot de passe en utilisant PBKDF2. Les algorithmes de dérivation de clé sont utilisés pour convertir un mot de passe en une clé de chiffrement de clé comme partie de la technique de gestion de clé basée sur mot de passe.

Les identifiants d'algorithme de dérivation de clé sont localisés dans les champs EnvelopedData RecipientInfos PasswordRecipientInfo keyDerivationAlgorithm et AuthenticatedData RecipientInfos PasswordRecipientInfo keyDerivationAlgorithm. La clé de chiffrement de clé qui est dérivée du mot de passe est utilisée pour chiffrer la clé de chiffrement de contenu.

Les clé de chiffrement de contenu chiffrés avec une clé de chiffrement de clé dérivé de mot de passe sont localisés dans le champ EnvelopedData RecipientInfos PasswordRecipientInfo encryptedKey. Les clés d'authentification de message chiffrés avec de clés de chiffrement de clé dérivés de mot de passe sont localisés dans le champ AuthenticatedData RecipientInfos PasswordRecipientInfo encryptedKey.

L'algorithme de dérivation de clé PBKDF2 est spécifié dans la rfc 2898. keyDerivationAlgorithmIdentifier identifie l'algorithme de dérivation de clé, et ses paramètres associés utilisés pour dériver la clé de chiffrement de clé du mot de passe fournis. L'identifiant d'algorithme pour l'algorithme de dérivation de clé PBKDF2 est :

```
id-PBKDF2 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-5(5) 12 }
```

Le champ parameter de AlgorithmIdentifier doit être PBKDF2-params :

```
PBKDF2-params ::= SEQUENCE {
 salt CHOICE {
  specified OCTET STRING,
  otherSource AlgorithmIdentifier },
 iterationCount INTEGER (1..MAX),
 keyLength INTEGER (1..MAX) OPTIONAL,
 prf AlgorithmIdentifier
  DEFAULT { algorithm hMAC-SHA1, parameters NULL } }
```

Dans le PBKDF2-params, le salt doit utiliser la chaîne d'octet spécifiée.

Algorithmes de chiffrement de contenu

Cette section spécifie les conventions employées par les implémentation CMS qui supportent le chiffrement de contenu en utilisant Triple-DES 3 clés en mode CBC, Triple 2-clés en mode CBC, ou RC2 en mode CBC.

Les identifiants d'algorithme de chiffrement de contenu sont localisés dans les champs EnvelopedData EncryptedContentInfo contentEncryptionAlgorithm et EncryptedData EncryptedContentInfo contentEncryptionAlgorithm. Les algorithme de chiffrement de contenu sont utilisé pour chiffrer le contenu localisé dans le champ EnvelopedData EncryptedContentInfo encryptedContent et le champ EncryptedData EncryptedContentInfo encryptedContent.

Triple-DES CBC

L'algorithme Triple-DES est composé de 3 opération DES séquentielles : chiffrer, déchiffrer, et chiffrer. Triple-DES 3-clés utilise une clé différente pour chaque opération DES. Triple-DES 2-clés utilise une clé pour les opérations de chiffrement et une clé différente pour l'opération de déchiffrement. Les même identifiant d'algorithme sont utilisé pour les 2.

L'identifiant d'algorithme pour Triple-DES en mode CBC est :

```
des-ede3-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) 7 }
```

Le champ parameters de AlgorithmIdentifier doit être présent, et le champ doit contenir un CBCParameter :

```
CBCParameter ::= IV
IV ::= OCTET STRING - exactement 8 octets
```

RC2 CBC

L'algorithme RC2 est décris dans la rfc2268. L'identifiant d'algorithme pour RC2 en mode CBC est :

```
rc2-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) 2 }
```

Le champ parameters de AlgorithmIdentifier doit être présent et doit contenir un RC2CBCParameter :

```
RC2CBCParameter ::= SEQUENCE {
 rc2ParameterVersion INTEGER,
 iv OCTET STRING } - exactement 8 octets
```

Les bits de clé effectifs RC2 (la taille de clé) supérieurs à 32 et inférieurs à 256 sont encodés dans le RC2ParameterVersion. Pour les tailles 40, 64, et 128, les valeurs RC2ParameterVersion sont 160, 120, et 58, respectivement. Ces valeurs ne sont pas simplement la longueur de clé RC2. Noter que la valeur 160 doit être encodé en 2 octets (00 A0), parce que l'octet A0 seul représente un nombre négatif.

Algorithms de code d'authentification de message

Cette section spécifie les conventions employées par les implémentations CMS qui supportent HMAC avec SHA-1. Les identifiant d'algorithme MAC sont localisés dans le champ macAlgorithm de AuthenticatedData. es valeurs MAC sont localisées dans le champ mac de AuthenticatedData.

HMAC avec SHA-1

L'algorithme HMAC avc SHA-1 est décris dans la rfc 2104. L'identifiant d'algorithme pour HMAC avec SHA-1 est :

```
hMAC-SHA1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) 8 1 2 }
```

Il y a 2 encodages possibles pour le champ parameters de AlgorithmIdentifier pour HMAC avec SHA-1. Les 2 alternatives surviennent du fait que quand la syntaxe 1988 pour AlgorithmIdentifier a été traduite en syntaxe 1997, l'optionnel associé avec les paramètres AlgortihmIdentifier ont été perdus. Plus tard, cet optionnel a été récupérée, mais jusque là beaucoup de gens pensaient que les paramètres de l'algorithme étaient obligatoires. À cause de cela certaines implémentations encodent les paramètres en éléments NUL et d'autres les omettent entièrement.

Le champ parameters de AlgorithmIdentifier est optionnel. Si présent, le champ parameters doit contenir NULL. Les implémentations doivent accepter HMAC avec SHA-1 avec parameters absent, et doivent accepter HMAC avec SHA-1 avec paramaters à NULL. Les implémentations devraient générer des AlgorithmIdentifiers HMAC avec SHA-1 avec parameters absent.

Module ASN.1

```
CryptographicMessageSyntaxAlgorithms
 { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cmsalg-2001(16) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN
 - EXPORTS All
 - Les types et valeurs définis dans ce module sont exporté pour l'utilisation dans d'autres modules ASN.1.
 - D'autres applications peuvent les utiliser.

IMPORTS
 - Imports de la rfc 3280, Appendix A.1
  AlgorithmIdentifier
   FROM PKIX1Explicit88 { iso(1)
    identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) mod(0)
pkix1-explicit(18) } ;

 - Algorithm Identifiers

 sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 26 }

 md5 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) digestAlgorithm(2) 5 }

 id-dsa OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 1 }

 id-dsa-with-sha1 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 3 }

 rsaEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 }

 md5WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
4 }

 sha1WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
5 }

 dh-public-number OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-x942(10046) number-type(2) 1 }
```

```
 id-alg-ESDH OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
alg(3) 5 }

 id-alg-SSDH OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
alg(3) 10 }

 id-alg-CMS3DESwrap OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) alg(3) 6 }

 id-alg-CMSRC2wrap OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) alg(3) 7 }

 des-ede3-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) 7 }

 rc2-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) 2 }

 hMAC-SHA1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) 8 1 2 }

 id-PBKDF2 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-5(5) 12 }

 - Public Key Types

 Dss-Pub-Key ::= INTEGER - Y

 RSAPublicKey ::= SEQUENCE {
  modulus INTEGER, - n
  publicExponent INTEGER } - e

 DHPublicKey ::= INTEGER - y = g^x mod p

 - Signature Value Types

 Dss-Sig-Value ::= SEQUENCE {
  r INTEGER,
  s INTEGER }

 - Algorithm Identifier Parameter Types

 Dss-Parms ::= SEQUENCE {
  p INTEGER,
  q INTEGER,
  g INTEGER }

 DHDomainParameters ::= SEQUENCE {
  p INTEGER, - odd prime, p=jq +1
  g INTEGER, - generator, g
  q INTEGER, - factor of p-1
  j INTEGER OPTIONAL, - subgroup factor
  validationParms ValidationParms OPTIONAL }

 ValidationParms ::= SEQUENCE {
  seed BIT STRING,
  pgenCounter INTEGER }

 KeyWrapAlgorithm ::= AlgorithmIdentifier

 RC2wrapParameter ::= RC2ParameterVersion
```

```
 RC2ParameterVersion ::= INTEGER

 CBCParameter ::= IV

 IV ::= OCTET STRING - exactly 8 octets

 RC2CBCParameter ::= SEQUENCE {
  rc2ParameterVersion INTEGER,
  iv OCTET STRING } - exactly 8 octets

 PBKDF2-params ::= SEQUENCE {
  salt CHOICE {
   specified OCTET STRING,
   otherSource AlgorithmIdentifier },
  iterationCount INTEGER (1..MAX),
  keyLength INTEGER (1..MAX) OPTIONAL,
  prf AlgorithmIdentifier
   DEFAULT { algorithm hMAC-SHA1, parameters NULL } }

END - of CryptographicMessageSyntaxAlgorithms
```

Considérations de sécurité

La technique de gestion de clé employée pour distribuer les clé d'authentification de message doit elle-même fournir l'authentification, sinon le contenu est délivré avec l'intégrité depuis une source inconnue. Ni RSA, ni Ephemeral-Static Diffie-Hellman ne fournissent l'authentification de l'origine des données nécessaires. Static-Static Diffie-Hellman fournis l'authentification de l'origine des données nécessaire quand l'auteur et les clés publiques des destinataires sont liés aux entités appropriés dans les certificats X.509.

Quand plus de 2 parties partagent la même clé d'authentification de message, l'authentification de l'origine des donnée n'est pas fournie. Toutes les parties qui connaissent la clé d'authentification de message peuvent calculer un MAC valide, donc le contenu pourrait venir de n'importe lequel de ces parties.

Les implémentations doivent générer des clé de chiffrement de contenu, de clé d'authentification de message, des vecteurs d'initialisation, des valeurs one-time aléatoirement, et du padding. Également, la génération de paires de clé publique/privées assure des nombres aléatoires. L'utilisation inadéquate de générateurs de nombre pseudo-aléatoires pour générer des valeurs peut engendrer des problèmes de sécurité. Un attaquant peut trouver qu'il est plus facile de reproduite l'environnement PRNG qui de produire les clé, en cherchant le plus petit jeu de possibilités résultants, au lieu de brute forcer tout l'espace de clé. La génération de nombre aléatoires de qualité est difficile. Les rfc1750 offre un guide important dans ce but, et l'appendice 3 de FIPS Pub 186 fournis une technique PRNG de qualité.

En utilisant les algorithmes d'agrément de clé ou des clé de chiffrement de clé symétrique précédemment distribués, une clé de chiffrement de clé est utilisée pour chiffrer la clé de chiffrement de contenu. Si les algorithmes de chiffrement de clé et de chiffrement de contenu sont différents, la sécurité effective est déterminée par le plus faible des 2. Les implémenteurs doivent s'assurer que les algorithmes de chiffrement de clé sont aussi fort sinon plus que les algorithmes de chiffrement de contenu.

La rfc 3217 spécifie des algorithmes d'enveloppe de clé utilisés pour chiffrer une clé de chiffrement de contenu Triple-DES avec une clé de chiffrement de clé Triple-DES ou pour chiffrer un clé de chiffrement de contenu RC2 avec une clé de chiffrement de clé RC2. Les algorithmes d'enveloppe de clé utilisent le mode CBC, et ont été revus pour l'utilisation avec Triple-DES et RC2. Ils n'ont pas été revus pour l'utilisation avec d'autres modes cryptographiques ou d'autres algorithmes de chiffrement. Cependant, si une implémentation CMS souhaite supporter d'autres chiffrements, les algorithmes d'enveloppe de clé additionnels doivent être définis pour supporter ces chiffrements additionnels.

Les implémenteurs devraient vérifier si les algorithmes cryptographiques deviennent faible dans le temps. De nouvelles techniques de cryptanalyse sont développées et les performances de calcul s'améliorent, Le facteur temps pour casser un algorithme cryptographique sera réduit. Cependant, les implémentations d'algorithmes crytographiques devraient être modulaires permettant aux nouveaux algorithmes

d'être facilement insérés.

Les utilisateurs de CMS, particulièrement ceux employant le CMS pour le support d'applications intéractives, devraient savoir que RSA, comme spécifié dans la rfc 2313, est vulnérable à des attaques de texte chiffré choisis lorsqu'il est appliqué à des fins de chiffrement. L'exploitation de cette vulnérabilité identifiée, révélant le résultat d'un déchiffrement RSA particulier, nécessite l'accès à un oracle qui répondra à un grand nombre de textes chiffrés (basé sur les résultats disponibles actuellement), qui sont construits adaptivement en réponse à une réponse reçue précédemment fournissant des informations sur la réussite ou l'échec des opération de déchiffrement tentés. En résultat, l'attaque apparaît significativement moins faisable dans les environnements S/MIME store-and-forward que pour les protocoles intéractif.

Une version mises à jours de PKCS#1 a été publiée, la version 2.0, qui préserve le support pour le format de padding de chiffrement, et définis une nouvelle alternative. Pour résoudre la vulnérabilité, la version 2.0 spécifie et recommande l'utilisation de OAEP quand RSA est utilisé pour fournir la confidentialité. Voir la rfc 3218 pour plus d'informations sur la vulnérabilité de PKCS#1 version 1.5.