S/MIME Working Group
Internet Draft
expires in six months

R. Housley
SPYRUS
May 1998

Cryptographic Message Syntax

<draft-ietf-smime-cms-05.txt>

Status of this Memo

Abstract

This document describes the Cryptographic Message Syntax. This syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary messages.

The Cryptographic Message Syntax is derived from PKCS #7 version 1.5 [RFC 2315]. Wherever possible, backward compatibility is preserved; however, changes were necessary to accommodate attribute certificate transfer and key agreement techniques for key management.

This draft is being discussed on the "ietf-smime" mailing list. To join the list, send a message to <ietf-smime-request@imc.org> with the single word "subscribe" in the body of the message. Also, there is a Web site for the mailing list at <http://www.imc.org/ietf-smime/>.

Acknowledgements

This document is the result of contributions from many professionals. I appreciate the hard work of all members of the IETF S/MIME Working Group. I extend a special thanks to Rich Ankney, Tim Dean, Steve Dusse, Paul Hoffman, Scott Hollenbeck, Burt Kaliski, John Pawling, Blake Ramsdell, Jim Schaad, and Dave Solo for their efforts and support.

1 Introduction

This document describes the Cryptographic Message Syntax. This syntax is used to digitally sign or encrypt arbitrary messages.

The Cryptographic Message Syntax describes an encapsulation syntax for data protection. It supports digital signatures and encryption. The syntax allows multiple encapsulation, so one encapsulation envelope can be nested inside another. Likewise, one party can digitally sign some previously encapsulated data. It also allows arbitrary attributes, such as signing time, to be signed along with the message content, and provides for other attributes such as countersignatures to be associated with a signature.

The Cryptographic Message Syntax can support a variety of architectures for certificate-based key management, such as the one defined by the PKIX working group.

The Cryptographic Message Syntax values are generated using ASN.1, using BER-encoding. Values are typically represented as octet strings. While many systems are capable of transmitting arbitrary octet strings reliably, it is well known that many electronic-mail systems are not. This document does not address mechanisms for encoding octet strings for reliable transmission in such environments.

2 General Overview

The Cryptographic Message Syntax is general enough to support many different content types. This document defines six content types: data, signed-data, enveloped-data, digested-data, encrypted-data, and authenticated-data. Also, additional content types can be defined outside this document.

An implementation that conforms to this specification must implement the data, signed-data, and enveloped-data content types. The other content types may be implemented if desired.

As a general design philosophy, content types permit single pass processing using indefinite-length Basic Encoding Rules (BER) encoding. Single-pass operation is especially helpful if content is large, stored on tapes, or is "piped" from another process. Single-pass operation has one significant drawback: it is difficult to perform encode operations using the Distinguished Encoding Rules (DER) encoding in a single pass since the lengths of the various components may not be known in advance. However, signed attributes within the signed-data content type and authenticated attributes within the authenticated-data content type require DER encoding. Signed attributes and authenticated attributes must be transmitted in DER form to ensure that recipients can validate a content that contains an unrecognized attribute.

3 General Syntax

The Cryptographic Message Syntax associates a protection content type with a protection content. The syntax shall have ASN.1 type ContentInfo:

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

ContentType ::= OBJECT IDENTIFIER
```

The fields of ContentInfo have the following meanings:

contentType indicates the type of protection content. It is an object identifier; it is a unique string of integers assigned by an authority that defines the content type.

content is the protection content. The type of protection content can be determined uniquely by contentType. Protection content types for signed-data, enveloped-data, digested-data, encrypted-data, and authenticated-data are defined in this document. If additional protection content types are defined in other documents, the ASN.1 type defined along with the object identifier should not be a CHOICE type.

4 Data Content Type

The following object identifier identifies the data content type:

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

The data content type is intended to refer to arbitrary octet

strings, such as ASCII text files; the interpretation is left to the application. Such strings need not have any internal structure (although they could have their own ASN.1 definition or other structure).

The data content type is generally used in conjunction with the signed-data, enveloped-data, digested-data, encrypted-data, and authenticated-data protection content types. The data content type is encapsulated in one of these protection content types.

5 Signed-data Content Type

The signed-data content type consists of a content of any type and zero or more signature values. Any number of signers in parallel can sign any type of content.

The typical application of the signed-data content type represents one signer's digital signature on content of the data content type. Another typical application disseminates certificates and certificate revocation lists (CRLs).

The process by which signed-data is constructed involves the following steps:

1. For each signer, a message digest, or hash value, is computed on the content with a signer-specific message-digest algorithm. If two signers employ the same message digest algorithm, then the message digest need be computed for only one of them. If the signer is signing any information other than the content, the message digest of the content and the other information are digested with the signer's message digest algorithm (see Section 5.4), and the result becomes the "message digest."

2. For each signer, the message digest is digitally signed using the signer's private key.

3. For each signer, the signature value and other signer-specific information are collected into a SignerInfo value, as defined in Section 5.3. Certificates and CRLs for each signer, and those not corresponding to any signer, are collected in this step.

4. The message digest algorithms for all the signers and the SignerInfo values for all the signers are collected together with the content into a SignedData value, as defined in Section 5.1.

A recipient independently computes the message digest. This message digest and the signer's public key are used to validate the signature value. The signer's public key is referenced by an issuer

distinguished name and an issuer-specific serial number that uniquely identify the certificate containing the public key. The signer's certificate may be included in the SignedData certificates field.

This section is divided into five parts. The first part describes the top-level type SignedData, the second part describes the per-signer information type SignerInfo, and the third, fourth, and fifth parts describe the message digest calculation, signature generation, and signature validation processes, respectively.

5.1 SignedData Type

The following object identifier identifies the signed-data content type:

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

The signed-data content type shall have ASN.1 type SignedData:

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

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

The fields of type SignedData have the following meanings:

version is the syntax version number. If no attribute certificates are present in the certificates field and the encapsulated content type is id-data, then the value of version shall be 1; however, if attribute certificates are present or the encapsulated content type is other than id-data, then the value of version shall be 3.

digestAlgorithms is a collection of message digest algorithm identifiers. There may be any number of elements in the collection, including zero. Each element identifies the message digest algorithm, along with any associated parameters, used by one or more signer. The collection is intended to list the message digest algorithms employed by all of the signers, in any order, to facilitate one-pass signature verification. The message

digesting process is described in Section 5.4.

encapContentInfo is the signed content, consisting of a content type identifier and the content itself. Details of the EncapsulatedContentInfo type are discussed in section 5.2.

certificates is a collection of certificates. It is intended that the set of certificates be sufficient to contain chains from a recognized "root" or "top-level certification authority" to all of the signers in the signerInfos field. There may be more certificates than necessary, and there may be certificates sufficient to contain chains from two or more independent top-level certification authorities. There may also be fewer certificates than necessary, if it is expected that recipients have an alternate means of obtaining necessary certificates (e.g., from a previous set of certificates). If no attribute certificates are present in the collection, then the value of version shall be 1; however, if attribute certificates are present, then the value of version shall be 3.

crls is a collection of certificate revocation lists (CRLs). It is intended that the set contain information sufficient to determine whether or not the certificates in the certificates field are valid, but such correspondence is not necessary. There may be more CRLs than necessary, and there may also be fewer CRLs than necessary.

signerInfos is a collection of per-signer information. There may be any number of elements in the collection, including zero. The details of the SignerInfo type are discussed in section 5.3.

The optional omission of the eContent within the EncapsulatedContentInfo field makes it possible to construct "external signatures." In the case of external signatures, the content being signed is absent from the EncapsulatedContentInfo value included in the signed-data content type. If the eContent value within EncapsulatedContentInfo is absent, then the signatureValue is calculated and the eContentType is assigned as though the eContent value was present.

In the degenerate case where there are no signers, the EncapsulatedContentInfo value being "signed" is irrelevant. In this case, the content type within the EncapsulatedContentInfo value being "signed" should be id-data (as defined in section 4), and the content field of the EncapsulatedContentInfo value should be omitted.

5.2 EncapsulatedContentInfo Type

Per-signer information is represented in the type SignerInfo:

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

ContentType ::= OBJECT IDENTIFIER
```

The fields of type EncapsulatedContentInfo have the following meanings:

eContentType is an object identifier uniquely specifies the content type.

eContent in the content itself, carried as an octet string. The eContent need not be DER encoded.

5.3 SignerInfo Type

Per-signer information is represented in the type SignerInfo:

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

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

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

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

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING
```

The fields of type SignerInfo have the following meanings:

version is the syntax version number; it shall be 1.

issuerAndSerialNumber specifies the signer's certificate (and thereby the signer's public key) by issuer distinguished name and issuer-specific serial number.

digestAlgorithm identifies the message digest algorithm, and any associated parameters, used by the signer. The message digest is computed over the encapsulated content and signed attributes, if present. The message digest algorithm should be among those listed in the digestAlgorithms field of the associated SignerData. The message digesting process is described in Section 5.4.

signedAttributes is a collection of attributes that are signed. The field is optional, but it must be present if the content type of the EncapsulatedContentInfo value being signed is not id-data. Each SignedAttribute in the SET must be DER encoded. Useful attribute types, such as signing time, are defined in Section 11. If the field is present, it must contain, at a minimum, the following two attributes:

A content-type attribute having as its value the content type of the EncapsulatedContentInfo value being signed. Section 11.1 defines the content-type attribute.

A message-digest attribute, having as its value the message digest of the content. Section 11.2 defines the message-digest attribute.

signatureAlgorithm identifies the signature algorithm, and any associated parameters, used by the signer to generate the digital signature.

signature is the result of digital signature generation, using the message digest and the signer's private key.

unsignedAttributes is a collection of attributes that are not signed. The field is optional. Useful attribute types, such as countersignatures, are defined in Section 11.

The fields of type SignedAttribute and UnsignedAttribute have the following meanings:

attrType indicates the type of attribute. It is an object identifier.

attrValues is a set of values that comprise the attribute. The type of each value in the set can be determined uniquely by attrType.

5.4 Message Digest Calculation Process

The message digest calculation process computes a message digest on either the content being signed or the content together with the signed attributes. In either case, the initial input to the message digest calculation process is the "value" of the encapsulated content being signed. Specifically, the initial input is the encapContentInfo eContent OCTET STRING to which the signing process is applied. Only the octets comprising the value of the eContent OCTET STRING are input to the message digest algorithm, not the tag or the length octets.

The result of the message digest calculation process depends on whether the signedAttributes field is present. When the field is absent, the result is just the message digest of the content as described above. When the field is present, however, the result is the message digest of the complete DER encoding of the SignedAttributes value contained in the signedAttributes field. Since the SignedAttributes value, when present, must contain the content type and the content message digest attributes, those values are indirectly included in the result. A separate encoding of the signedAttributes field is performed for message digest calculation. The IMPLICIT [0] tag in the signedAttributes field is not used for the DER encoding, rather an EXPLICIT SET OF tag is used. That is, the DER encoding of the SET OF tag, rather than of the IMPLICIT [0] tag, is to be included in the message digest calculation along with the length and content octets of the SignedAttributes value.

When the signedAttributes field is absent, then only the octets comprising the value of the signedData encapContentInfo eContent OCTET STRING (e.g., the contents of a file) are input to the message digest calculation. This has the advantage that the length of the content being signed need not be known in advance of the signature generation process.

Although the encapContentInfo eContent OCTET STRING tag and length octets are not included in the message digest calculation, they are still protected by other means. The length octets are protected by the nature of the message digest algorithm since it is computationally infeasible to find any two distinct messages of any length that have the same message digest.

5.5 Message Signature Generation Process

The input to the signature generation process includes the result of the message digest calculation process and the signer's private key. The details of the signature generation depend on the signature algorithm employed. The object identifier, along with any

parameters, that specifies the signature algorithm employed by the signer is carried in the signatureAlgorithm field. The signature value generated by the signer is encoded as an OCTET STRING and carried in the signature field.

5.6 Message Signature Validation Process

The input to the signature validation process includes the result of the message digest calculation process and the signer's public key. The details of the signature validation depend on the signature algorithm employed.

The recipient may not rely on any message digest values computed by the originator. If the signedData signerInfo includes signedAttributes, then the content message digest must be calculated as described in section 5.4. For the signature to be valid, the message digest value calculated by the recipient must be the same as the value of the messageDigest attribute included in the signedAttributes of the signedData signerInfo.

6 Enveloped-data Content Type

The enveloped-data content type consists of an encrypted content of any type and encrypted content-encryption keys for one or more recipients. The combination of the encrypted content and one encrypted content-encryption key for a recipient is a "digital envelope" for that recipient. Any type of content can be enveloped for an arbitrary number of recipients.

The typical application of the enveloped-data content type will represent one or more recipients' digital envelopes on content of the data or signed-data content types.

Enveloped-data is constructed by the following steps:

1. A content-encryption key for a particular content-encryption algorithm is generated at random.

2. The content-encryption key is encrypted for each recipient. The details of this encryption depend on the key management algorithm used, but three general techniques are supported:

   key transport: the content-encryption key is encrypted in the recipient's public key;

   key agreement: the recipient's public key and the sender's private key are used to generate a pairwise symmetric key, then the content-encryption key is encrypted in the pairwise

symmetric key; and

mail list keys: the content-encryption key is encrypted in a previously distributed symmetric key.

3. For each recipient, the encrypted content-encryption key and other recipient-specific information are collected into a RecipientInfo value, defined in Section 6.2.

4. The content is encrypted with the content-encryption key. Content encryption may require that the content be padded to a multiple of some block size; see Section 6.3.

5. The RecipientInfo values for all the recipients are collected together with the encrypted content to form an EnvelopedData value as defined in Section 6.1.

A recipient opens the digital envelope by decrypting one of the encrypted content-encryption keys and then decrypting the encrypted content with the recovered content-encryption key.

This section is divided into four parts. The first part describes the top-level type EnvelopedData, the second part describes the per-recipient information type RecipientInfo, and the third and fourth parts describe the content-encryption and key-encryption processes.

6.1 EnvelopedData Type

The following object identifier identifies the enveloped-data content type:

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

The enveloped-data content type shall have ASN.1 type EnvelopedData:

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

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

RecipientInfos ::= SET OF RecipientInfo
```

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

EncryptedContent ::= OCTET STRING
```

The fields of type EnvelopedData have the following meanings:

version is the syntax version number. If originatorInfo is present, then version shall be 2. If any of the RecipientInfo structures included have a version other than 0, then the version shall be 2. If originatorInfo is absent and all of the RecipientInfo structures are version 0, then version shall be 0.

originatorInfo optionally provides information about the originator. It is present only if required by the key management algorithm. It may contain certificates and CRLs:

certs is a collection of certificates. certs may contain originator certificates associated with several different key management algorithms. The certificates contained in certs are intended to be sufficient to make chains from a recognized "root" or "top-level certification authority" to all recipients. However, certs may contain more certificates than necessary, and there may be certificates sufficient to make chains from two or more independent top-level certification authorities. Alternatively, certs may contain fewer certificates than necessary, if it is expected that recipients have an alternate means of obtaining necessary certificates (e.g., from a previous set of certificates).

crls is a collection of CRLs. It is intended that the set contain information sufficient to determine whether or not the certificates in the certs field are valid, but such correspondence is not necessary. There may be more CRLs than necessary, and there may also be fewer CRLs than necessary.

recipientInfos is a collection of per-recipient information. There must be at least one element in the collection.

encryptedContentInfo is the encrypted content information.

The fields of type EncryptedContentInfo have the following meanings:

contentType indicates the type of content.

contentEncryptionAlgorithm identifies the content-encryption

algorithm, and any associated parameters, used to encrypt the content. The content-encryption process is described in Section 6.3. The same algorithm is used for all recipients.

encryptedContent is the result of encrypting the content. The field is optional, and if the field is not present, its intended value must be supplied by other means.

The recipientInfos field comes before the encryptedContentInfo field so that an EnvelopedData value may be processed in a single pass.

6.2 RecipientInfo Type

Per-recipient information is represented in the type RecipientInfo. RecipientInfo has a different format for the three key management techniques that are supported: key transport, key agreement, and previously distributed mail list keys. In all cases, the content-encryption key is transferred to one or more recipient in encrypted form.

```
RecipientInfo ::= CHOICE {
  ktri KeyTransRecipientInfo,
  kari KeyAgreeRecipientInfo,
  mlri MailListRecipientInfo }

EncryptedKey ::= OCTET STRING
```

6.2.1 KeyTransRecipientInfo Type

Per-recipient information using key transport is represented in the type KeyTransRecipientInfo. Each instance of KeyTransRecipientInfo transfers the content-encryption key to one recipient.

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

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

The fields of type KeyTransRecipientInfo have the following meanings:

version is the syntax version number. If the RecipientIdentifier is the CHOICE issuerAndSerialNumber, then the version shall be 0. If the RecipientIdentifier is rKeyId, then the version shall be 2.

rid specifies the recipient's certificate or key that was used by the sender to protect the content-encryption key.

keyEncryptionAlgorithm identifies the key-encryption algorithm, and any associated parameters, used to encrypt the content-encryption key for the recipient. The key-encryption process is described in Section 6.4.

encryptedKey is the result of encrypting the content-encryption key for the recipient.

The EntityIdentifier is a CHOICE with two alternatives specifying the recipient's certificate, and thereby the recipient's public key. The recipient's certificate must contain a key transport public key. The content-encryption key is encrypted with the recipient's public key. The issuerAndSerialNumber alternative identifies the recipient's certificate by the issuer's distinguished name and the certificate serial number; the subjectKeyIdentifier identifies the recipient's certificate by the X.509 subjectKeyIdentifier extension value.

6.2.2 KeyAgreeRecipientInfo Type

Recipient information using key agreement is represented in the type KeyAgreeRecipientInfo. Each instance of KeyAgreeRecipientInfo will transfer the content-encryption key to one or more recipient.

```
KeyAgreeRecipientInfo := SEQUENCE {
  version Version,  -- always set to 3
  originatorCert [0] EXPLICIT EntityIdentifier,
  ukm [1] EXPLICIT UserKeyingMaterial OPTIONAL,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  recipientEncryptedKeys RecipientEncryptedKeys }

RecipientEncryptedKeys ::= SEQUEENCE OF RecipientEncryptedKey

RecipientEncryptedKey := SEQUENCE {
  rid RecipientIdentifier,
  encryptedKey EncryptedKey }

RecipientIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  rKeyId [0] IMPLICIT RecipientKeyIdentifier }

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

```
SubjectKeyIdentifier ::= OCTET STRING
```

The fields of type KeyAgreeRecipientInfo have the following meanings:

version is the syntax version number. It shall always be 3.

originatorCert is a CHOICE with two alternatives specifying the sender's certificate, and thereby the sender's public key. The sender's certificate must contain a key agreement public key, and the sender uses the corresponding private key and the recipient's public key to generate a pairwise key. The content-encryption key is encrypted in the pairwise key. The issuerAndSerialNumber alternative identifies the sender's certificate by the issuer's distinguished name and the certificate serial number; the subjectKeyIdentifier alternative identifies the sender's certificate by the X.509 subjectKeyIdentifier extension value.

ukm is optional. With some key agreement algorithms, the sender provides a User Keying Material (UKM) to ensure that a different key is generated each time the same two parties generate a pairwise key.

keyEncryptionAlgorithm identifies the key-encryption algorithm, and any associated parameters, used to encrypt the content-encryption key in the key-encryption key. The key-encryption process is described in Section 6.4.

recipientEncryptedKeys includes a recipient identifier and the encrypted key for one or more recipients. The RecipientIdentifier is a CHOICE with two alternatives specifying the recipient's certificate, and thereby the recipient's public key, that was used by the sender to generate a pairwise key. The recipient's certificate must contain a key agreement public key. The content-encryption key is encrypted in the pairwise key. The issuerAndSerialNumber alternative identifies the recipient's certificate by the issuer's distinguished name and the certificate serial number; the RecipientKeyIdentifier is described below. The encryptedKey is the result of encrypting the content-encryption key in the pairwise key generated using the key agreement algorithm.

The fields of type RecipientKeyIdentifier have the following meanings:

subjectKeyIdentifier identifies the recipient's certificate by the X.509 subjectKeyIdentifier extension value.

date is optional. When present, the date specifies which of the

recipient's previously distributed UKMs was used by the sender.

other is optional. When present, this field contains additional information used by the recipient to locate the public keying material used by the sender.

6.2.3 MailListRecipientInfo Type

Recipient information using previously distributed symmetric keys is represented in the type MailListRecipientInfo. Each instance of MailListRecipientInfo will transfer the content-encryption key to one or more recipients who have the previously distributed key-encryption key.

```
MailListRecipientInfo := SEQUENCE {
  version Version,  -- always set to 4
  mlid MailListKeyIdentifier,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey }

MailListKeyIdentifier ::= SEQUENCE {
  kekIdentifier OCTET STRING,
  date GeneralizedTime OPTIONAL,
  other OtherKeyAttribute OPTIONAL }
```

The fields of type MailListRecipientInfo have the following meanings:

version is the syntax version number. It shall always be 4.

mlKeyId specifies a symmetric key encryption key that was previously distributed to the sender and one or more recipients.

keyEncryptionAlgorithm identifies the key-encryption algorithm, and any associated parameters, used to encrypt the content-encryption key in the key-encryption key. The key-encryption process is described in Section 6.4.

encryptedKey is the result of encrypting the content-encryption key in the key-encryption key.

The fields of type MailListKeyIdentifier have the following meanings:

kekIdentifier identifies the key-encryption key that was previously distributed to the sender and one or more recipients.

date is optional. When present, the date specifies a single key-encryption key from a set that was previously distributed.

other is optional. When present, this field contains additional information used by the recipient to determine the key-encryption key used by the sender.

6.3 Content-encryption Process

The content-encryption key for the desired content-encryption algorithm is randomly generated. The data to be protected is padded as described below, then the padded data is encrypted using the content-encryption key. The encryption operation maps an arbitrary string of octets (the data) to another string of octets (the ciphertext) under control of a content-encryption key. The encrypted data is included in the envelopedData encryptedContentInfo encryptedContent OCTET STRING.

The input to the content-encryption process is the "value" of the content being enveloped. Only the value octets of the envelopedData encryptedContentInfo encryptedContent OCTET STRING are encrypted; the OCTET STRING tag and length octets are not encrypted.

Some content-encryption algorithms assume the input length is a multiple of k octets, where k is greater than one. For such algorithms, the input shall be padded at the trailing end with k-(l mod k) octets all having value k-(l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:

```
                01 -- if l mod k = k-1
             02 02 -- if l mod k = k-2
                 .
                 .
                 .
      k k ... k k -- if l mod k = 0
```

The padding can be removed unambiguously since all input is padded, including input values that are already a multiple of the block size, and no padding string is a suffix of another. This padding method is well defined if and only if k is less than 256.

6.4 Key-encryption Process

The input to the key-encryption process -- the value supplied to the recipient's key-encryption algorithm --is just the "value" of the content-encryption key.

7 Digested-data Content Type

The digested-data content type consists of content of any type and a message digest of the content.

Typically, the digested-data content type is used to provide content integrity, and the result generally becomes an input to the enveloped-data content type.

The following steps construct digested-data:

1. A message digest is computed on the content with a message-digest algorithm.

2. The message-digest algorithm and the message digest are collected together with the content into a DigestedData value.

A recipient verifies the message digest by comparing the message digest to an independently computed message digest.

The following object identifier identifies the digested-data content type:

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

The digested-data content type shall have ASN.1 type DigestedData:

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

Digest ::= OCTET STRING
```

The fields of type DigestedData have the following meanings:

version is the syntax version number. It shall be 0.

digestAlgorithm identifies the message digest algorithm, and any associated parameters, under which the content is digested. The message-digesting process is the same as in Section 5.4 in the case when there are no signed attributes.

encapContentInfo is the content that is digested, as defined in section 5.2.

digest is the result of the message-digesting process.

The ordering of the digestAlgorithm field, the encapContentInfo field, and the digest field makes it possible to process a DigestedData value in a single pass.

8 Encrypted-data Content Type

The encrypted-data content type consists of encrypted content of any type. Unlike the enveloped-data content type, the encrypted-data content type has neither recipients nor encrypted content-encryption keys. Keys must be managed by other means.

The typical application of the encrypted-data content type will be to encrypt the content of the data content type for local storage, perhaps where the encryption key is a password.

The following object identifier identifies the encrypted-data content type:

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

The encrypted-data content type shall have ASN.1 type EncryptedData:

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

The fields of type EncryptedData have the following meanings:

version is the syntax version number. It shall be 0.

encryptedContentInfo is the encrypted content information, as defined in Section 6.1.

9 Authenticated-data Content Type

The authenticated-data content type consists of content of any type, a message authentication code (MAC), and encrypted authentication keys for one or more recipients. The combination of the MAC and one encrypted authentication key for a recipient is necessary for that recipient to validate the integrity of the content. Any type of content can be integrity protected for an arbitrary number of recipients.

The process by which authenticated-data is constructed involves the following steps:

1. A message-authentication key for a particular message-authentication algorithm is generated at random.

2. The message-authentication key is encrypted for each recipient. The details of this encryption depend on the key management algorithm used.

3. For each recipient, the encrypted message-authentication key and other recipient-specific information are collected into a RecipientInfo value, defined in Section 6.2.

4. Using the message-authentication key, the originator computes a MAC value on the content. If the originator is authenticating any information in addition to the content (see Section 9.2), the MAC value of the content and the other information are generated using the same message authentication code algorithm and key, and the result becomes the "MAC value."

9.1 AuthenticatedData Type

The following object identifier identifies the authenticated-data content type:

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

The authenticated-data content type shall have ASN.1 type AuthenticatedData:

```
AuthenticatedData ::= SEQUENCE {
  version Version,
  originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
  recipientInfos RecipientInfos,
  macAlgorithm MessageAuthenticationCodeAlgorithm,
  encapContentInfo EncapsulatedContentInfo,
  authenticatedAttributes [1] IMPLICIT AuthAttributes OPTIONAL,
  mac MessageAuthenticationCode,
  unauthenticatedAttributes [2] IMPLICIT UnauthAttributes OPTIONAL }

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

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

MessageAuthenticationCode ::= OCTET STRING
```

The fields of type AuthenticatedData have the following meanings:

version is the syntax version number. It shall be 0.

originatorInfo optionally provides information about the originator. It is present only if required by the key management algorithm. It may contain certificates, CRLs, and user keying material (UKMs), as defined in Section 6.1.

recipientInfos is a collection of per-recipient information, as defined in Section 6.1. There must be at least one element in the collection.

macAlgorithm is a message authentication code algorithm identifier. It identifies the message authentication code algorithm, along with any associated parameters, used by the originator. Placement of the macAlgorithm field facilitates one-pass processing by the recipient.

encapContentInfo is the content that is authenticated, as defined in section 5.2.

authenticatedAttributes is a collection of attributes that are authenticated. The field is optional, but it must be present if the content type of the EncapsulatedContentInfo value being authenticated is not id-data. Each AuthenticatedAttribute in the SET must be DER encoded. Useful attribute types are defined in Section 11. If the field is present, it must contain, at a minimum, the following two attributes:

A content-type attribute having as its value the content type of the EncapsulatedContentInfo value being signed. Section 11.1 defines the content-type attribute.

A mac-value attribute, having as its value the message authentication code of the content. Section 11.5 defines the mac-value attribute.

mac is the message authentication code.

unauthenticatedAttributes is a collection of attributes that are not authenticated. The field is optional. Useful attribute types are defined in Section 11.

9.2 MAC Generation

The MAC calculation process computes a message authentication code on either the message content or the content together with the originator's authenticated attributes.

If there are no authenticated attributes, the MAC input data is the content octets of the DER encoding of the content field of the ContentInfo value to which the MAC process is applied. Only the contents octets of the DER encoding of that field are input to the MAC algorithm, not the identifier octets or the length octets.

If authenticated attributes are present, they must include the content-type attribute (as described in Section 11.1) and mac-value attribute (as described in section 11.5). The MAC input data is the complete DER encoding of the Attributes value contained in the authenticatedAttributes field. Since the Attributes value, when the field is present, must contain as attributes the content type and the mac value of the content, those values are indirectly included in the result. A separate encoding of the authenticatedAttributes field is performed for MAC calculation. The IMPLICIT [0] tag in the authenticatedAttributes field is not used for the DER encoding, rather an EXPLICIT SET OF tag is used. That is, the DER encoding of the SET OF tag, rather than of the IMPLICIT [0] tag, is to be included in the MAC calculation along with the length and contents octets of the AuthAttributes value.

If the content has content type id-data and the authenticatedAttributes field is absent, then just the value of the data (e.g., the contents of a file) is input to the MAC calculation. This has the advantage that the length of the content need not be known in advance of the MAC calculation process. Although the tag and length octets are not included in the MAC calculation, they are still protected by other means. The length octets are protected by the nature of the MAC algorithm since it is computationally infeasible to find any two distinct messages of any length that have the same MAC.

The fact that the MAC is computed on part of a DER encoding does not mean that DER is the required method of representing that part for data transfer. Indeed, it is expected that some implementations will store objects in forms other than their DER encodings, but such practices do not affect MAC computation.

The input to the MAC calculation process includes the MAC input data, defined above, and an authentication key conveyed in a recipientInfo structure. The details of MAC calculation depend on the MAC algorithm employed (e.g., DES-MAC and HMAC). The object identifier, along with any parameters, that specifies the MAC algorithm employed by the originator is carried in the macAlgorithm field. The MAC value generated by the originator is encoded as an OCTET STRING and carried in the mac field.

9.3 MAC Validation

The input to the MAC validation process includes the input data (determined based on the presence or absence of authenticated attributes, as defined in 9.2), and the authentication key conveyed in recipientInfo. The details of the MAC validation process depend on the MAC algorithm employed.

The recipient may not rely on any MAC values computed by the originator. If the originator includes authenticated attributes, then the content of the authenticatedAttributes must be autenticated as described in section 9.2. For the MAC to be valid, the message MAC value calculated by the recipient must be the same as the value of the macValue attribute included in the authenticatedAttributes. Likewise, the attribute MAC value calculated by the recipient must be the same as the value of the mac field included in the authenticatedData.

10 Useful Types

This section is divided into two parts. The first part defines algorithm identifiers, and the second part defines other useful types.

10.1 Algorithm Identifier Types

All of the algorithm identifiers have the same type: AlgorithmIdentifier. The definition of AlgorithmIdentifier is imported from X.509.

There are many alternatives for each type of algorithm listed. For each of these five types, Section 12 lists the algorithms that must be included in a CMS implementation.

10.1.1 DigestAlgorithmIdentifier

The DigestAlgorithmIdentifier type identifies a message-digest algorithm. Examples include SHA-1, MD2, and MD5. A message-digest algorithm maps an octet string (the message) to another octet string (the message digest).

```
DigestAlgorithmIdentifier ::= AlgorithmIdentifier
```

10.1.2 SignatureAlgorithmIdentifier

The SignatureAlgorithmIdentifier type identifies a signature algorithm. Examples include DSS and RSA. A signature algorithm supports signature generation and verification operations. The

signature generation operation uses the message digest and the signer's private key to generate a signature value. The signature verification operation uses the message digest and the signer's public key to determine whether or not a signature value is valid. Context determines which operation is intended.

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

10.1.3 KeyEncryptionAlgorithmIdentifier

The KeyEncryptionAlgorithmIdentifier type identifies a key-encryption algorithm used to encrypt a content-encryption key. The encryption operation maps an octet string (the key) to another octet string (the encrypted key) under control of a key-encryption key. The decryption operation is the inverse of the encryption operation. Context determines which operation is intended.

The details of encryption and decryption depend on the key management algorithm used. Key transport, key agreement, and previously distributed symmetric key-encrypting keys are supported.

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

10.1.4 ContentEncryptionAlgorithmIdentifier

The ContentEncryptionAlgorithmIdentifier type identifies a content-encryption algorithm. Examples include DES, Triple-DES, and RC2. A content-encryption algorithm supports encryption and decryption operations. The encryption operation maps an octet string (the message) to another octet string (the ciphertext) under control of a content-encryption key. The decryption operation is the inverse of the encryption operation. Context determines which operation is intended.

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

10.1.5 MessageAuthenticationCodeAlgorithm

The MessageAuthenticationCodeAlgorithm type identifies a message authentication code (MAC) algorithm. Examples include DES MAC and HMAC. A MAC algorithm supports generation and verification operations. The MAC generation and verification operations use the same symmetric key. Context determines which operation is intended.

MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier

10.2 Other Useful Types

This section defines types that are used other places in the document. The types are not listed in any particular order.

10.2.1 CertificateRevocationLists

The CertificateRevocationLists type gives a set of certificate revocation lists (CRLs). It is intended that the set contain information sufficient to determine whether the certificates with which the set is associated are revoked or not. However, there may be more CRLs than necessary or there may be fewer CRLs than necessary.

The definition of CertificateList is imported from X.509.

```
CertificateRevocationLists ::= SET OF CertificateList
```

10.2.2 CertificateChoices

The CertificateChoices type gives either a PKCS #6 extended certificate [PKCS #6], an X.509 certificate, or an X.509 attribute certificate. The PKCS #6 extended certificate is obsolete. It is included for backward compatibility, and its use should be avoided.

The definitions of Certificate and AttributeCertificate are imported from X.509.

```
CertificateChoices ::= CHOICE {
  certificate Certificate,  -- See X.509
  extendedCertificate [0] IMPLICIT ExtendedCertificate,  -- Obsolete
  attrCert [1] IMPLICIT AttributeCertificate }  -- See X.509 and X9.57
```

10.2.3 CertificateSet

The CertificateSet type provides a set of certificates. It is intended that the set be sufficient to contain chains from a recognized "root" or "top-level certification authority" to all of the sender certificates with which the set is associated. However, there may be more certificates than necessary, or there may be fewer than necessary.

The precise meaning of a "chain" is outside the scope of this document. Some applications may impose upper limits on the length of a chain; others may enforce certain relationships between the subjects and issuers of certificates within a chain.

```
CertificateSet ::= SET OF CertificateChoices
```

10.2.4 IssuerAndSerialNumber

The IssuerAndSerialNumber type identifies a certificate, and thereby an entity and a public key, by the distinguished name of the certificate issuer and an issuer-specific certificate serial number.

The definition of Name is imported from X.501, and the definition of CertificateSerialNumber is imported from X.509.

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

CertificateSerialNumber ::= INTEGER
```

10.2.5 Version

The Version type gives a syntax version number, for compatibility with future revisions of this document.

```
Version ::= INTEGER  { v0(0), v1(1), v2(2), v3(3), v4(4) }
```

10.2.6 UserKeyingMaterial

The UserKeyingMaterial type gives a syntax user keying material (UKM). Some key agreement algorithms require UKMs to ensure that a different key is generated each time the same two parties generate a pairwise key. The sender provides a UKM for use with a specific key agreement algorithm.

```
UserKeyingMaterial ::= OCTET STRING
```

10.2.7 OtherKeyAttribute

The OtherKeyAttribute type gives a syntax for the inclusion of other key attributes that permit the recipient to select the key used by the sender. The attribute object identifier must be registered along with the syntax of the attribute itself. Use of this structure should be avoided since it may impede interoperability.

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

11 Useful Attributes

This section defines attributes that may used with signed-data or authenticated-data. Some of these attributes were originally defined

in PKCS #9 [PKCS #9], others are defined and specified here. The attributes are not listed in any particular order.

11.1 Content Type

The content-type attribute type specifies the content type of the ContentInfo value being signed in signed-data. The content-type attribute type is required if there are any authenticated attributes present.

The content-type attribute must be a signed attribute or an authenticated attribute; it cannot be an unsigned attribute or unauthenticated attribute.

The following object identifier identifies the content-type attribute:

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

Content-type attribute values have ASN.1 type ContentType:

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

A content-type attribute must have a single attribute value.

11.2 Message Digest

The message-digest attribute type specifies the message digest of the encapContentInfo eContent OCTET STRING being signed in signed-data (see section 5.4), where the message digest is computed using the signer's message digest algorithm.

Within signed-data, the message-digest signed attribute type is required if there are any attributes present.

The message-digest attribute must be a signed attribute; it cannot be an unsigned attribute, an authenticated attribute, or unauthenticated attribute.

The following object identifier identifies the message-digest attribute:

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

Message-digest attribute values have ASN.1 type MessageDigest:

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

A message-digest attribute must have a single attribute value.

11.3 Signing Time

The signing-time attribute type specifies the time at which the signer (purportedly) performed the signing process. The signing-time attribute type is intended for use in signed-data.

The signing-time attribute may be a signed attribute; it cannot be an unsigned attribute, an authenticated attribute, or an unauthenticated attribute.

The following object identifier identifies the signing-time attribute:

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

Signing-time attribute values have ASN.1 type SigningTime:

```
SigningTime ::= Time

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

Note: The definition of Time matches the one specified in the 1997 version of X.509.

Dates through the year 2049 must be encoded as UTCTime, and dates in the year 2050 or later must be encoded as GeneralizedTime.

A signing-time attribute must have a single attribute value.

No requirement is imposed concerning the correctness of the signing time, and acceptance of a purported signing time is a matter of a recipient's discretion. It is expected, however, that some signers, such as time-stamp servers, will be trusted implicitly.

11.4 Countersignature

The countersignature attribute type specifies one or more signatures on the contents octets of the DER encoding of the signatureValue field of a SignerInfo value in signed-data. Thus, the countersignature attribute type countersigns (signs in serial) another signature.

The countersignature attribute must be an unsigned attribute; it cannot be a signed attribute, an authenticated attribute, or an unauthenticated attribute.

The following object identifier identifies the countersignature attribute:

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

Countersignature attribute values have ASN.1 type Countersignature:

```
Countersignature ::= SignerInfo
```

Countersignature values have the same meaning as SignerInfo values for ordinary signatures, except that:

1. The signedAttributes field must contain a message-digest attribute if it contains any other attributes, but need not contain a content-type attribute, as there is no content type for countersignatures.

2. The input to the message-digesting process is the contents octets of the DER encoding of the signatureValue field of the SignerInfo value with which the attribute is associated.

A countersignature attribute can have multiple attribute values.

The fact that a countersignature is computed on a signature value means that the countersigning process need not know the original content input to the signing process. This has advantages both in efficiency and in confidentiality. A countersignature, since it has type SignerInfo, can itself contain a countersignature attribute. Thus it is possible to construct arbitrarily long series of countersignatures.

11.5 Message Authentication Code (MAC) Value

The MAC-value attribute type specifies the MAC of the encapContentInfo eContent OCTET STRING being authenticated in authenticated-data (see section 9), where the MAC value is computed using the originator's MAC algorithm and the data-authentication key.

Within authenticated-data, the MAC-value attribute type is required if there are any authenticated attributes present.

The MAC-value attribute must be a authenticated attribute; it cannot be an signed attribute, an unsigned attribute, or unauthenticated

attribute.

The following object identifier identifies the MAC-value attribute:

```
id-macValue OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) aa(2) 8 }
```

MAC-value attribute values have ASN.1 type MACValue:

```
MACValue ::= OCTET STRING
```

A MAC-value attribute must have a single attribute value.

12 Supported Algorithms

This section lists the algorithms that must be implemented. Additional algorithms that may be implemented are also included.

12.1 Digest Algorithms

CMS implementations must include SHA-1. CMS implementations may include MD5.

12.1.1 SHA-1

[*** Add pointer to algorithm specification. Provide OID. ***]

12.1.2 MD5

[*** Add pointer to algorithm specification. Provide OID. ***]

12.2 Signature Algorithms

CMS implementations must include DSA. CMS implementations may include RSA.

12.2.1 DSA

[*** Add pointer to algorithm specification. Provide OID. Provide ASN.1 for parameters and signature value. ***]

12.2.2 RSA

[*** Add pointer to algorithm specification. Provide OID. Provide ASN.1 for parameters and signature value. ***]

12.3 Key Encryption Algorithms

CMS implementations must include X9.42 Static Diffie-Hellman. CMS implementations may include RSA and Triple-DES.

12.3.1 X9.42 Static Diffie-Hellman

[*** Add pointer to algorithm specification. Provide OID. Provide ASN.1 for parameters. ***]

12.3.2 RSA

[*** Add pointer to algorithm specification. Provide OID. Provide ASN.1 for parameters. ***]

12.3.3 Triple-DES Key Wrap

[*** Add pointer to algorithm specification. Provide OID. ***]

12.4 Content Encryption Algorithms

CMS implementations must include Triple-DES in CBC mode. CMS implementations may include DES in CBC mode and RC2 in CBC mode.

12.4.1 Triple-DES CBC

[*** Add pointer to algorithm specification. Provide OID. ***]

12.4.2 DES CBC

[*** Add pointer to algorithm specification. Provide OID. ***]

12.4.3 RC2 CBC

[*** Add pointer to algorithm specification. Provide OID. ***]

12.5 Message Authentication Code Algorithms

No MAC algorithms are mandatory. CMS implementations may include DES MAC and HMAC.

12.5.1 DES MAC

[*** Add pointer to algorithm specification. Provide OID. ***]

12.5.2 HMAC

[*** Add pointer to algorithm specification. Provide OID. ***]

Appendix A: ASN.1 Module

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

   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   -- EXPORTS All --
   -- The types and values defined in this module are exported for use in
   -- the other ASN.1 modules.  Other applications may use them for their
   -- own purposes.

   IMPORTS

     -- Directory Information Framework (X.501)
           Name
              FROM InformationFramework { joint-iso-itu-t ds(5) modules(1)
                   informationFramework(1) 3 }

     -- Directory Authentication Framework (X.509)
           AlgorithmIdentifier, AttributeCertificate, Certificate,
           CertificateList, CertificateSerialNumber
              FROM AuthenticationFramework { joint-iso-itu-t ds(5)
                   module(1) authenticationFramework(7) 3 } ;


   -- Cryptographic Message Syntax

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

   ContentType ::= OBJECT IDENTIFIER

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

   DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

   SignerInfos ::= SET OF SignerInfo
```

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

   ContentType ::= OBJECT IDENTIFIER

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

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

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

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

   AttributeValue ::= ANY

   SignatureValue ::= OCTET STRING

   EnvelopedData ::= SEQUENCE {
     version Version,
     originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
     recipientInfos RecipientInfos,
     encryptedContentInfo EncryptedContentInfo }

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

   RecipientInfos ::= SET OF RecipientInfo

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

   EncryptedContent ::= OCTET STRING
```

```
RecipientInfo ::= CHOICE {
  ktri KeyTransRecipientInfo,
  kari KeyAgreeRecipientInfo,
  mlri MailListRecipientInfo }

EncryptedKey ::= OCTET STRING

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

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

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

RecipientEncryptedKeys ::= SEQUEENCE OF RecipientEncryptedKey

RecipientEncryptedKey := SEQUENCE {
  rid RecipientIdentifier,
  encryptedKey EncryptedKey }

RecipientIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  rKeyId [0] IMPLICIT RecipientKeyIdentifier }

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

SubjectKeyIdentifier ::= OCTET STRING

MailListRecipientInfo := SEQUENCE {
  version Version,  -- always set to 4
  mlid MailListKeyIdentifier,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey }
```

```
MailListKeyIdentifier ::= SEQUENCE {
  kekIdentifier OCTET STRING,
  date GeneralizedTime OPTIONAL,
  other OtherKeyAttribute OPTIONAL }

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

Digest ::= OCTET STRING

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

AuthenticatedData ::= SEQUENCE {
  version Version,
  originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
  recipientInfos RecipientInfos,
  macAlgorithm MessageAuthenticationCodeAlgorithm,
  encapContentInfo EncapsulatedContentInfo,
  authenticatedAttributes [1] IMPLICIT AuthAttributes OPTIONAL,
  mac MessageAuthenticationCode,
  unauthenticatedAttributes [2] IMPLICIT UnauthAttributes OPTIONAL }

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

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

MessageAuthenticationCode ::= OCTET STRING

DigestAlgorithmIdentifier ::= AlgorithmIdentifier

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier

CertificateRevocationLists ::= SET OF CertificateList
```

```
   CertificateChoices ::= CHOICE {
     certificate Certificate,  -- See X.509
     extendedCertificate [0] IMPLICIT ExtendedCertificate,  -- Obsolete
     attrCert [1] IMPLICIT AttributeCertificate }  -- See X.509 & X9.57

   CertificateSet ::= SET OF CertificateChoices

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

   KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

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

   UserKeyingMaterial ::= OCTET STRING

   UserKeyingMaterials ::= SET SIZE (1..MAX) OF UserKeyingMaterial

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


   -- CMS Attributes

   MessageDigest ::= OCTET STRING

   SigningTime  ::= Time

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

   Countersignature ::= SignerInfo

   MACValue ::= OCTET STRING
```

```
   -- Object Identifiers

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

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

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

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

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

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

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

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

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

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

   id-macValue OBJECT IDENTIFIER ::= { iso(1) member-body(2)
       us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) aa(2) 8 }


   -- Obsolete Extended Certificate syntax from PKCS#6

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

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

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

   Signature ::= BIT STRING


   END -- of CryptographicMessageSyntax
```

References

RFC 2313 Kaliski, B. PKCS #1: RSA Encryption, Version 1.5. March 1998.

RFC 2315 Kaliski, B. PKCS #7: Cryptographic Message Syntax, Version 1.5. March 1998.

PKCS #6 RSA Laboratories. PKCS #6: Extended-Certificate Syntax Standard, Version 1.5. November 1993.

PKCS #9 RSA Laboratories. PKCS #9: Selected Attribute Types, Version 1.1. November 1993.

X.208 CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). 1988.

X.209 CCITT. Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). 1988.

X.501 CCITT. Recommendation X.501: The Directory - Models. 1988.

X.509 CCITT. Recommendation X.509: The Directory - Authentication Framework. 1988.

Security Considerations

The Cryptographic Message Syntax provides a method for digitally signing data, digesting data, encrypting data, and authenticating data.

Implementations must protect the signer's private key. Compromise of the signer's private key permits masquerade.

Implementations must protect the key management private key and the content-encryption key. Compromise of the key management private key may result in the disclosure of all messages protected with that key. Similarly, compromise of the content-encryption key may result in disclosure of the encrypted content.

Author Address

Russell Housley
SPYRUS
381 Elden Street
Suite 1120
Herndon, VA 20170
USA

housley@spyrus.com