Cryptographic Message Syntax

<draft-ietf-smime-cms-03.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. 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/>.

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 authenticated 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.

The Cryptographic Message Syntax exports one content type, ContentInfo, as well as the various object identifiers.

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. Since the signed-data content type requires DER encoding, an extra pass may be necessary when a content type other than data is encapsulated.

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 authenticating any information other than the content (see Section 5.2), the message digest of the content and the other information are digested with the signer's message digest algorithm, 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 5.2. 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

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

ContentType ::= OBJECT IDENTIFIER

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, then the value of version shall be 1; however, if attribute certificates are present, 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 5.3.

encapContentInfo is the content that is signed. It is a sequence of a content type identifier and the content itself. An object identifier uniquely specifies the content type. The content itself is carried in an octet string.

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 optional omission of the encapContentInfo field makes it possible to construct "external signatures." In the case of external signatures, the content being signed would be absent from the EncapsulatedContentInfo value included in the signed-data content type. If the EncapsulatedContentInfo value is absent, the signatureValue is calculated as though the EncapsulatedContentInfo value was present. The presumed EncapsulatedContentInfo must have the content type set to id-data (as defined in section 4) and the content omitted.

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 data (as defined in section 4), and the content field of the EncapsulatedContentInfo value should be omitted.

5.2 SignerInfo Type

Per-signer information is represented in the type SignerInfo:

```
SignerInfo ::= SEQUENCE {
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm DigestAlgorithmIdentifier,
  authenticatedAttributes [0] IMPLICIT CMSAttributes OPTIONAL,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature SignatureValue,
  unauthenticatedAttributes [1] IMPLICIT CMSAttributes OPTIONAL }

CMSAttributes ::= SET OF CMSAttribute

CMSAttribute ::= SEQUENCE {
  cmsAttrType OBJECT IDENTIFIER,
  critical BOOLEAN DEFAULT FALSE,
  cmsAttrValues SET OF CMSAttributeValue }

CMSAttributeValue ::= ANY

SignatureValue ::= OCTET STRING
```

The fields of type SignerInfo have the following meanings:

version is the syntax version number. If any of the authenticated attributes, are critical, then the version shall be 3. If all of the authenticated attributes are non-critical, then the version shall be 1. If the authenticatedAttributes and field is absent, then version 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 authenticated attributes, if present. The message digest algorithm should be among those listed in the digestAlgorithms field of the associated SignerInfo value. The message digesting process is described in Section 5.3.

authenticatedAttributes 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 data. The field may include critical and non-critical

attributes. 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.

unauthenticatedAttributes is a collection of attributes that are not signed. The field is optional, and it may not include critical attributes. Useful attribute types, such as countersignatures, are defined in Section 11.

The fields of type CMSAttribute have the following meanings:

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

critical is a boolean value. TRUE indicates that the attribute is critical, and FALSE indicates that the attribute is non-critical. A recipient must reject the signed-data if it encounters a critical attribute that it does not recognize; however, an unrecognized non-critical attribute may be ignored.

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

5.3 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 signer's authenticated 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 content OCTET STRING of the content field of the EncapsulatedContentInfo value to which the signing process is

applied. Only the contents of the OCTET STRING are input to the message digest algorithm, not the identifier octets or the length octets.

The result of the message digest calculation process depends on whether the authenticatedAttributes 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 Attributes value contained in the authenticatedAttributes field. Since the Attributes value, when present, must contain as attributes the content type and the content message digest, those values are indirectly included in the result. A separate encoding of the authenticatedAttributes field is performed for message digest 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 message digest calculation along with the length and content octets of the CMSAttributes value.

When the content being signed has a content type of data (as defined in section 4) and the authenticatedAttributes field is absent, then just the value of the data (e.g., the contents of a file) is 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 identifier octets and the 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.

The fact that the message digest 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 message digest computation.

5.4 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.

5.5 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 authenticatedAttributes, then content message digest must be calculated as described in section 5.3. 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 authenticatedAttributes 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,
  ukms [2] IMPLICIT UserKeyingMaterials 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 of 2, 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, CRLs, and user keying material (UKMs):

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.

ukms is a collection of UKMs. The set includes a UKM for each key management algorithm employed by the originator that requires one. In general, several recipients will use each UKM in the set.

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 ::= SEQUENCE {
  version Version,
  rid RecipientIdentifier,
  originatorCert [0] EXPLICIT EntityIdentifier OPTIONAL,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey }

RecipientIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  rKeyId [0] IMPLICIT RecipientKeyIdentifier,
  mlKeyId [1] IMPLICIT MailListKeyIdentifier }

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

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

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

SubjectKeyIdentifier ::= OCTET STRING
```

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

The fields of type RecipientInfo have the following meanings:

version is the syntax version number. If the OriginatorCert is absent and the RecipientIdentifier is the CHOICE issuerAndSerialNumber, then the version shall be 0. If the OriginatorCert is present or the RecipientIdentifier is either the CHOICE rKeyId or mlKeyId, 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.

originatorCert optionally specifies the originator's certificate to be used by this recipient. This field should be included when the originator has more than one certificate containing a public key associated with the key management algorithm used for this recipient.

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 RecipientIdentifier is a CHOICE with three alternatives. The first two alternatives, issuerAndSerialNumber and rKeyId, specifies the recipient's certificate, and thereby the recipient's public key. The rKeyId alternative may optionally specify other parameters needed, such as the date. If the recipient's certificate contains a key transport public key, then the content-encryption key is encrypted with the recipient's public key. If the recipient's certificate contains a key agreement public key, then a pairwise symmetric key is established and used to encrypt the content-encryption key. The third alternative, mlKeyId, specifies a symmetric key encryption key that was previously distributed to the sender and recipient.

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 UKMs was used by the sender.

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

The fields of type MailListKeyIdentifier have the following meanings:

kekIdentifier identifies the key-encryption key that was previously distributed to the sender and the recipient.

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

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

6.3 Content-encryption Process

The input to the content-encryption process is the "value" of the content being enveloped. Only the content octets; identifier or length octets are not included.

When the content being enveloped has content type of data (as defined in section 4), then just the value of the data (e.g., the contents of a file) is encrypted. This has the advantage that the length of the content being encrypted need not be known in advance of the encryption process.

The identifier octets and the length octets are not encrypted. The length octets may be protected implicitly by the encryption process, depending on the encryption algorithm. The identifier octets are not protected at all, although they can be recovered from the content type, assuming that the content type uniquely determines the identifier octets. Explicit protection of the identifier and length octets requires that the signed-data content type be employed prior to digital enveloping.

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 5.3 in the case when there are no authenticated attributes.

encapContentInfo is the content that is digested, as defined in section 5.1.

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 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,
  mac MessageAuthenticationCode }

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 5.1.

mac is the message authentication code.

10 Useful Types

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

10.1 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 ContentEncryptionAlgorithmIdentifier

The ContentEncryptionAlgorithmIdentifier type identifies a content-encryption algorithm such as DES. 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.

The definition of AlgorithmIdentifier is imported from X.509.

```
ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
```

10.3 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).

The definition of AlgorithmIdentifier is imported from X.509.

```
DigestAlgorithmIdentifier ::= AlgorithmIdentifier
```

10.4 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.

The definition of AlgorithmIdentifier is imported from X.509.

```
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
```

10.5 CertificateChoices

The CertificateChoices type gives either a PKCS #6 extended certificate, 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
```

10.6 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
```

10.7 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 SerialNumber is imported from X.509.

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

SerialNumber ::= INTEGER
```

10.8 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.

The definition of AlgorithmIdentifier is imported from X.509.

```
KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
```

10.9 Version

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

```
Version ::= INTEGER
```

10.10 UserKeyingMaterial

The UserKeyingMaterial type gives a syntax user keying material (UKM). Some key management algorithms require UKMs. The sender provides a UKM for the specific key management algorithm. The UKM is employed by all of the recipients that use the same key encryption algorithm.

```
UserKeyingMaterial ::= SEQUENCE {
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  ukm OCTET STRING }
```

10.11 UserKeyingMaterials

The UserKeyingMaterial type provides a set of user keying materials (UKMs). This allows the sender to provide a UKM for each key management algorithm that requires one.

```
UserKeyingMaterials ::= SET OF UserKeyingMaterial
```

10.12 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 }
```

10.13 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.

The definition of AlgorithmIdentifier is imported from X.509.

```
MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier
```

11 Useful Attributes

This section defines attributes that may used with signed-data. All of these attributes were originally defined in PKCS #9, and they are included here for easy reference. 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 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 contents octets of the DER encoding of the content field of the ContentInfo value being signed in signed-data, where the message digest is computed using the signer's message digest algorithm. The message-digest attribute type is required if there are any authenticated attributes present.

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 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 unauthenticated attribute; it cannot be an authenticated 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 authenticatedAttributes 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.

12 Supported Algorithms

To be supplied. However, this section will list the must implement algorithms and other algorithms that may be implemented. It will include:

MUST implement: DSS, SHA-1, Diffie-Hellman (X9.42), and Triple-DES CBC (with three keys).

MAY implement: RSA (signature and key management), MD5, RC2 (40 bit), DES CBC, and DES MAC.

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

   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

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

   SignerInfos ::= SET OF SignerInfo
```

```
SignerInfo ::= SEQUENCE {
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm DigestAlgorithmIdentifier,
  authenticatedAttributes [0] IMPLICIT CMSAttributes OPTIONAL,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature SignatureValue,
  unauthenticatedAttributes [1] IMPLICIT CMSAttributes OPTIONAL }

CMSAttributes ::= SET OF CMSAttribute

CMSAttribute ::= SEQUENCE {
  cmsAttrType OBJECT IDENTIFIER,
  critical BOOLEAN DEFAULT FALSE,
  cmsAttrValues SET OF CMSAttributeValue }

CMSAttributeValue ::= 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,
  ukms [2] IMPLICIT UserKeyingMaterials OPTIONAL }

RecipientInfos ::= SET OF RecipientInfo

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

EncryptedContent ::= OCTET STRING

RecipientInfo ::= SEQUENCE {
  version Version,
  rid RecipientIdentifier,
  originatorCert [0] EXPLICIT EntityIdentifier OPTIONAL,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey }
```

```
RecipientIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  rKeyId [0] IMPLICIT RecipientKeyIdentifier,
  mlKeyId [1] IMPLICIT MailListKeyIdentifier }

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

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

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

SubjectKeyIdentifier ::= OCTET STRING

EncryptedKey ::= OCTET STRING

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,
  mac MessageAuthenticationCode }

MessageAuthenticationCode ::= OCTET STRING

CertificateRevocationLists ::= SET OF CertificateList

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
```

```
DigestAlgorithmIdentifier ::= AlgorithmIdentifier

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

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

CertificateSet ::= SET OF CertificateChoices

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

SerialNumber ::= INTEGER

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

Version ::= INTEGER

UserKeyingMaterial ::= SEQUENCE {
  algorithm AlgorithmIdentifier,
  ukm OCTET STRING }

UserKeyingMaterials ::= SET OF UserKeyingMaterial

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

MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier


-- CMS Attributes

MessageDigest ::= OCTET STRING

SigningTime  ::= Time

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

Countersignature ::= SignerInfo
```

```
-- 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 }
```

```
-- 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 Attributes }

Signature ::= BIT STRING


END -- of CryptographicMessageSyntax
```

References

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

PKCS #7 RSA Laboratories. PKCS #7: Cryptographic Message Syntax Standard. Version 1.5, November 1993.

PKCS #7: Cryptographic Message Syntax, Internet Draft draft-hoffman-pkcs-crypt-msg-xx.

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
PO Box 1198
Herndon, VA 20172
USA
housley@spyrus.com