PKCS #7: Cryptographic Message Syntax
Version 1.5

Status of this Memo

Copyright Notice

Overview

   This document describes a general syntax for data that may have
   cryptography applied to it, such as digital signatures and digital
   envelopes. The syntax admits recursion, so that, for example, one
   envelope can be nested inside another, or one party can sign some
   previously enveloped digital data. It also allows arbitrary
   attributes, such as signing time, to be authenticated along with the
   content of a message, and provides for other attributes such as
   countersignatures to be associated with a signature. A degenerate
   case of the syntax provides a means for disseminating certificates
   and certificate-revocation lists.

1. Scope

   This document is compatible with Privacy-Enhanced Mail (PEM) in that
   signed-data and signed-and-enveloped-data content, constructed in a
   PEM-compatible mode, can be converted into PEM messages without any
   cryptographic operations. PEM messages can similarly be converted
   into the signed-data and signed-and-enveloped data content types.

   This document can support a variety of architectures for
   certificate-based key management, such as the one proposed for
   Privacy-Enhanced Mail in RFC 1422. Architectural decisions such as
   what certificate issuers are considered "top-level," what entities
   certificate issuers are authorized to certify, what distinguished
   names are considered acceptable, and what policies certificate
   issuers must follow (such as signing only with secure hardware, or
   requiring entities to present specific forms of identification) are
   left outside the document.

The values produced according to this document are intended to be
BER-encoded, which means that the values would typically be
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 as (say) strings of ASCII
characters or other techniques for enabling reliable transmission by
re-encoding the octet string. RFC 1421 suggests one possible solution
to this problem.

2. References

    FIPS PUB 46-1 National Bureau of Standards. FIPS PUB 46-1:
              Data Encryption Standard. January 1988.

    PKCS #1   RSA Laboratories. PKCS #1: RSA Encryption.
              Version 1.5, November 1993.

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

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

    RFC 1421   Linn, J., "Privacy Enhancement for
              Internet Electronic Mail: Part I: Message
              Encryption and Authentication Procedures," RFC 1421
              February 1993.

    RFC 1422   Kent, S., "Privacy Enhancement for
              Internet Electronic Mail: Part II: Certificate-
              Based Key Management," RFC 1422, February 1993.

    RFC 1423   Balenson, D., "Privacy Enhancement for
              Internet Electronic Mail: Part III: Algorithms,
              Modes, and Identifiers," RFC 1423, February 1993.

    RFC 1424   Kaliski, B., "Privacy Enhancement for
              Internet Electronic Mail: Part IV: Key
              Certification and Related Services," RFC 1424,
              February 1993.

      RFC 1319   Kaliski, B., "The MD2 Message-Digest
                 Algorithm," RFC 1319, April 1992.

      RFC 1321   Rivest, R., "The MD5 Message-Digest
                 Algorithm," RFC 1321, April 1992.

      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.500   CCITT. Recommendation X.500: The Directory--
                 Overview of Concepts, Models and
                 Services. 1988.

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

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

      [NIST91]   NIST. Special Publication 500-202: Stable
                 Implementation Agreements for Open Systems
                 Interconnection Protocols. Version 5, Edition 1,
                 Part 12. December 1991.

      [RSA78]   R.L. Rivest, A. Shamir, and L. Adleman. A method
                 for obtaining digital signatures and public-key
                 cryptosystems. Communications of the ACM,
                 21(2):120-126, February 1978.

3. Definitions

   For the purposes of this document, the following definitions apply.

   AlgorithmIdentifier: A type that identifies an algorithm (by object
   identifier) and associated parameters. This type is defined in X.509.

   ASN.1: Abstract Syntax Notation One, as defined in X.208.

   Attribute: A type that contains an attribute type (specified by
   object identifier) and one or more attribute values. This type is
   defined in X.501.

   BER: Basic Encoding Rules, as defined in X.209.

Certificate: A type that binds an entity's distinguished name to a public key with a digital signature. This type is defined in X.509. This type also contains the distinguished name of the certificate issuer (the signer), an issuer-specific serial number, the issuer's signature algorithm identifier, and a validity period.

CertificateSerialNumber: A type that uniquely identifies a certificate (and thereby an entity and a public key) among those signed by a particular certificate issuer. This type is defined in X.509.

CertificateRevocationList: A type that contains information about certificates whose validity an issuer has prematurely revoked. The information consists of an issuer name, the time of issue, the next scheduled time of issue, and a list of certificate serial numbers and their associated revocation times. The CRL is signed by the issuer. The type intended by this document is the one defined RFC 1422.

DER: Distinguished Encoding Rules for ASN.1, as defined in X.509, Section 8.7.

DES: Data Encryption Standard, as defined in FIPS PUB 46-1.

desCBC: The object identifier for DES in cipher-block chaining (CBC) mode, as defined in [NIST91].

ExtendedCertificate: A type that consists of an X.509 public-key certificate and a set of attributes, collectively signed by the issuer of the X.509 public-key certificate. This type is defined in PKCS #6.

MD2: RSA Data Security, Inc.'s MD2 message-digest algorithm, as defined in RFC 1319.

md2: The object identifier for MD2, as defined in RFC 1319.

MD5: RSA Data Security, Inc.'s MD5 message-digest algorithm, as defined in RFC 1321.

md5: The object identifier for MD5, as defined in RFC 1321.

Name: A type that uniquely identifies or "distinguishes" objects in an X.500 directory. This type is defined in X.501. In an X.509 certificate, the type identifies the certificate issuer and the entity whose public key is certified.

PEM: Internet Privacy-Enhanced Mail, as defined in RFCs 1421-1424.

RSA: The RSA public-key cryptosystem, as defined in [RSA78].

rsaEncryption: The object identifier for RSA encryption, as defined in PKCS #1.

4. Symbols and abbreviations

No symbols or abbreviations are defined in this document.

5. General overview

The following nine sections specify useful types, general syntax, six content types, and object identifiers.

The syntax is general enough to support many different content types. This document defines six: data, signed data, enveloped data, signed-and-enveloped data, digested data, and encrypted data. Other content types may be added in the future. The use of content types defined outside this document is possible, but is subject to bilateral agreement between parties exchanging content.

This document exports one type, ContentInfo, as well as the various object identifiers.

There are two classes of content types: base and enhanced. Content types in the base class contain "just data," with no cryptographic enhancements. Presently, one content type is in this class, the data content type. Content types in the enhanced class contain content of some type (possibly encrypted), and other cryptographic enhancements. For example, enveloped-data content can contain (encrypted) signed-data content, which can contain data content. The four non-data content types fall into the enhanced class. The content types in the enhanced class thus employ encapsulation, giving rise to the terms "outer" content (the one containing the enhancements) and "inner" content (the one being enhanced).

The document is designed such that the enhanced content types can be prepared in a single pass using indefinite-length BER encoding, and processed in a single pass in any BER encoding. Single-pass operation is especially helpful if content is stored on tapes, or is "piped" from another process. One of the drawbacks of single-pass operation, however, is that it is difficult to output a DER encoding in a single pass, since the lengths of the various components may not be known in advance. Since DER encoding is required by the signed-data, signed-and-enveloped data, and digested-data content types, an extra pass may be necessary when a content type other than data is the inner content of one of those content types.

6. Useful types

   This section defines types that are useful in at least two places in
   the document.

6.1 CertificateRevocationLists

   The CertificateRevocationLists type gives a set of certificate-
   revocation lists. It is intended that the set contain information
   sufficient to determine whether the certificates with which the set
   is associated are "hot listed," but there may be more certificate-
   revocation lists than necessary, or there may be fewer than
   necessary.

   CertificateRevocationLists ::=
     SET OF CertificateRevocationList

6.2 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.

   ContentEncryptionAlgorithmIdentifier ::=
     AlgorithmIdentifier

6.3 DigestAlgorithmIdentifier

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

   DigestAlgorithmIdentifier ::= AlgorithmIdentifier

6.4 DigestEncryptionAlgorithmIdentifier

   The DigestEncryptionAlgorithmIdentifier type identifies a digest-
   encryption algorithm under which a message digest can be encrypted.
   One example is PKCS #1's rsaEncryption. A digest-encryption algorithm
   supports encryption and decryption operations. The encryption
   operation maps an octet string (the message digest) to another octet
   .bp string (the encrypted message digest) under control of a digest-
   encryption key. The decryption operation is the inverse of the

encryption operation. Context determines which operation is intended.

```
DigestEncryptionAlgorithmIdentifier ::=
  AlgorithmIdentifier
```

6.5 ExtendedCertificateOrCertificate

The ExtendedCertificateOrCertificate type gives either a PKCS #6 extended certificate or an X.509 certificate. This type follows the syntax recommended in Section 6 of PKCS #6:

```
ExtendedCertificateOrCertificate ::= CHOICE {
  certificate Certificate, -- X.509

  extendedCertificate [0] IMPLICIT ExtendedCertificate }
```

6.6 ExtendedCertificatesAndCertificates

The ExtendedCertificatesAndCertificates type gives a set of extended certificates and X.509 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 signers with which the set is associated, but there may be more certificates than necessary, or there may be fewer than necessary.

```
ExtendedCertificatesAndCertificates ::=
  SET OF ExtendedCertificateOrCertificate
```

Note. The precise meaning of a "chain" is outside the scope of this document. Some applications of this document may impose upper limits on the length of a chain; others may enforce certain relationships between the subjects and issuers of certificates in a chain. An example of such relationships has been proposed for Privacy-Enhanced Mail in RFC 1422.

6.7 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.

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

6.8 KeyEncryptionAlgorithmIdentifier

   The KeyEncryptionAlgorithmIdentifier type identifies a key-encryption
   algorithm under which a content-encryption key can be encrypted. One
   example is PKCS #1's rsaEncryption. A key-encryption algorithm
   supports encryption and decryption operations. 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.

   KeyEncryptionAlgorithmIdentifier ::=
     AlgorithmIdentifier

6.9 Version

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

   Version ::= INTEGER

7. General syntax

   The general syntax for content exchanged between entities according
   to this document associates a content type with content. The syntax
   shall have ASN.1 type ContentInfo:

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

   ContentType ::= OBJECT IDENTIFIER

   The fields of type ContentInfo have the following meanings:

        o   contentType indicates the type of content. It is
              an object identifier, which means it is a unique string of
              integers assigned by the authority that defines the content
              type. This document defines six content types (see Section
              14): data, signedData, envelopedData,
              signedAndEnvelopedData, digestedData, and encryptedData.

        o   content is the content. The field is optional, and
              if the field is not present, its intended value must be
              supplied by other means. Its type is defined along with the
              object identifier for contentType.

Notes.

    1.   The methods below assume that the type of content
can be determined uniquely by contentType, so the type
defined along with the object identifier should not be a
CHOICE type.

    2.   When a ContentInfo value is the inner content of
signed-data, signed-and-enveloped-data, or digested-data
content, a message-digest algorithm is applied to the
contents octets of the DER encoding of the content field.
When a ContentInfo value is the inner content of
enveloped-data or signed-and-enveloped-data content, a
content-encryption algorithm is applied to the contents
octets of a definite-length BER encoding of the content
field.

    3.   The optional omission of the content field makes
it possible to construct "external signatures," for
example, without modification to or replication of the
content to which the signatures apply. In the case of
external signatures, the content being signed would be
omitted from the "inner" encapsulated ContentInfo value
included in the signed-data content type.

8. Data content type

The data content type is just an octet string. It shall have ASN.1
type Data:

    Data ::= OCTET STRING

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 may; they could even be DER encodings).

9. Signed-data content type

The signed-data content type consists of content of any type and
encrypted message digests of the content for zero or more signers.
The encrypted digest for a signer is a "digital signature" on the
content for that signer. Any type of content can be signed by any
number of signers in parallel. Furthermore, the syntax has a
degenerate case in which there are no signers on the content. The
degenerate case provides a means for disseminating certificates and
certificate-revocation lists.

It is expected that the typical application of the signed-data
content type will be to represent one signer's digital signature on
content of the data content type. Another typical application will be
to disseminate certificates and certificate-revocation lists.

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

    1.   For each signer, a message digest 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 9.2), 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 and associated
         information are encrypted with the signer's private key.

    3.   For each signer, the encrypted message digest and
         other signer-specific information are collected into a
         SignerInfo value, defined in Section 9.2. Certificates and
         certificate-revocation lists 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, defined
         in Section 9.1.

A recipient verifies the signatures by decrypting the encrypted
message digest for each signer with the signer's public key, then
comparing the recovered message digest to an independently computed
message digest. The signer's public key is either contained in a
certificate included in the signer information, or is referenced by
an issuer distinguished name and an issuer-specific serial number
that uniquely identify the certificate for the public key.

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 and fourth parts describe
the message-digesting and digest-encryption processes. The fifth part
summarizes compatibility with Privacy-Enhanced Mail.

9.1 SignedData type

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

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

   DigestAlgorithmIdentifiers ::=

     SET OF DigestAlgorithmIdentifier

   SignerInfos ::= SET OF SignerInfo

   The fields of type SignedData have the following meanings:

         o   version is the syntax version number. It shall be
               1 for this version of the document.

         o   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
               (and any associated parameters) under which the
               content is digested for a some 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 9.3.

         o   contentInfo is the content that is signed. It can
               have any of the defined content types.

         o   certificates is a set of PKCS #6 extended
               certificates and X.509 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
               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
                those verifying the signatures have an alternate means of
                obtaining necessary certificates (e.g., from a previous set
                of certificates).

        o   crls is a set of certificate-revocation lists. It
                is intended that the set contain information sufficient to
                determine whether or not the certificates in the
                certificates field are "hot listed," but such
                correspondence is not necessary. There may be more
                certificate-revocation lists than necessary, and there may
                also be fewer certificate-revocation lists than necessary.

        o   signerInfos is a collection of per-signer
                information. There may be any number of elements in the
                collection, including zero.

    Notes.

        1.   The fact that the digestAlgorithms field comes
                before the contentInfo field and the signerInfos field
                comes after it makes it possible to process a SignedData
                value in a single pass. (Single-pass processing is
                described in Section 5.)

        2.   The differences between version 1 SignedData and
                version 0 SignedData (defined in PKCS #7, Version 1.4) are
                the following:

                o   the digestAlgorithms and signerInfos
                        fields may contain zero elements in version 1,
                        but not in version 0

                o   the crls field is allowed in version 1,
                        but not in version 0

                Except for the difference in version number, version 0
                SignedData values are acceptable as version 1 values. An
                implementation can therefore process SignedData values of
                either version as though they were version 1 values. It is
                suggested that PKCS implementations generate only version 1
                SignedData values, but be prepared to process SignedData
                values of either version.

3.   In the degenerate case where there are no signers
     on the content, the ContentInfo value being "signed" is
     irrelevant. It is recommended in that case that the content
     type of the ContentInfo value being "signed" be data, and
     the content field of the ContentInfo value be omitted.

9.2 SignerInfo type

   Per-signer information is represented in the type SignerInfo:

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

   EncryptedDigest ::= OCTET STRING

   The fields of type SignerInfo have the following meanings:

        o   version is the syntax version number. It shall be
             1 for this version of the document.

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

        o   digestAlgorithm identifies the message-digest
             algorithm (and any associated parameters) under which the
             content and authenticated attributes (if present) are
             digested. It should be among those in the digestAlgorithms
             field of the superior SignerInfo value. The message-
             digesting process is described in Section 9.3.

        o   authenticatedAttributes is a set of attributes
             that are signed (i.e., authenticated) by the signer. The
             field is optional, but it must be present if the content
             type of the ContentInfo value being signed is not data. If
             the field is present, it must contain, at a minimum, two
             attributes:

1.   A PKCS #9 content-type attribute having
      as its value the content type of the
      ContentInfo value being signed.

2.   A PKCS #9 message-digest attribute,
      having as its value the message digest
      of the content (see below).

Other attribute types that might be useful here, such as
signing time, are also defined in PKCS #9.

o   digestEncryptionAlgorithm identifies the digest-
     encryption algorithm (and any associated parameters) under
     which the message digest and associated information are
     encrypted with the signer's private key. The digest-
     encryption process is described in Section 9.4.

o   encryptedDigest is the result of encrypting the
     message digest and associated information with the signer's
     private key.

o   unauthenticatedAttributes is a set of attributes
     that are not signed (i.e., authenticated) by the signer.
     The field is optional. Attribute types that might be useful
     here, such as countersignatures, are defined in PKCS #9.

Notes.

1.   It is recommended in the interest of PEM
      compatibility that the authenticatedAttributes field be
      omitted whenever the content type of the ContentInfo value
      being signed is data and there are no other authenticated
      attributes.

2.   The difference between version 1 SignerInfo and
      version 0 SignerInfo (defined in PKCS #7, Version 1.4) is
      in the message-digest encryption process (see Section 9.4).
      Only the PEM-compatible processes are different, reflecting
      changes in Privacy-Enhanced Mail signature methods. There
      is no difference in the non-PEM-compatible message-digest
      encryption process.

      It is suggested that PKCS implementations generate only
      version 1 SignedData values. Since the PEM signature method
      with which version 0 is compatible is obsolescent, it is
      suggested that PKCS implementations be prepared to receive
      only version 1 SignedData values.

9.3 Message-digesting process

   The message-digesting 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-digesting process is the "value" of the content being signed.
   Specifically, the initial input is the contents octets of the DER
   encoding of the content field of the ContentInfo value to which the
   signing process is applied. Only the contents octets of the DER
   encoding of that field are digested, not the identifier octets or the
   length octets.

   The result of the message-digesting process (which is called,
   informally, the "message digest") depends on whether the
   authenticatedAttributes field is present. When the field is absent,
   the result is just the message digest of the content. When the field
   is present, however, the result is the message digest of the complete
   DER encoding of the Attributes value containted in the
   authenticatedAttributes field. (For clarity: The IMPLICIT [0] tag in
   the authenticatedAttributes field is not part of the Attributes
   value. The Attributes value's tag is SET OF, and the DER encoding of
   the SET OF tag, rather than of the IMPLICIT [0] tag, is to be
   digested along with the length and contents octets of the Attributes
   value.) Since the Attributes value, when the field is present, must
   contain as attributes the content type and the message digest of the
   content, those values are indirectly included in the result.

   When the content being signed has content type data and the
   authenticatedAttributes field is absent, then just the value of the
   data (e.g., the contents of a file) is digested. This has the
   advantage that the length of the content being signed need not be
   known in advance of the encryption process. This method is compatible
   with Privacy-Enhanced Mail.

   Although the identifier octets and the length octets are not
   digested, they are still protected by other means. The length octets
   are protected by the nature of the message-digest algorithm since it
   is by assumption computationally infeasible to find any two distinct
   messages of any length that have the same message digest.
   Furthermore, assuming that the content type uniquely determines the
   identifier octets, the identifier octets are protected implicitly in
   one of two ways: either by the inclusion of the content type in the
   authenticated attributes, or by the use of the PEM-compatible
   alternative in Section 9.4 which implies that the content type is
   data.

Note. 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 of this document may store objects in other than
their DER encodings, but such practices do not affect message-digest
computation.

9.4 Digest-encryption process

The input to the digest-encryption process--the value supplied to the
signer's digest-encryption algorithm--includes the result of the
message-digesting process (informally, the "message digest") and the
digest algorithm identifier (or object identifier). The result of the
digest-encryption process is the encryption with the signer's private
key of the BER encoding of a value of type DigestInfo:

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

Digest ::= OCTET STRING
```

The fields of type DigestInfo have the following meanings:

    o   digestAlgorithm identifies the message-digest
          algorithm (and any associated parameters) under which the
          content and authenticated attributes are digested. It
          should be the same as the digestAlgorithm field of the
          superior SignerInfo value.

    o   digest is the result of the message-digesting
          process.

Notes.

    1.   The only difference between the signature process
          defined here and the signature algorithms defined in PKCS
          #1 is that signatures there are represented as bit strings,
          for consistency with the X.509 SIGNED macro. Here,
          encrypted message digests are octet strings.

    2.   The input to the encryption process typically will
          have 30 or fewer octets. If digestEncryptionAlgorithm is
          PKCS #1's rsaEncryption, then this means that the input can
          be encrypted in a single block as long as the length of the
          RSA modulus is at least 328 bits, which is reasonable and
          consistent with security recommendations.

   3.   A message-digest algorithm identifier is included
        in the DigestInfo value to limit the damage resulting from
        the compromise of one message-digest algorithm. For
        instance, suppose an adversary were able to find messages
        with a given MD2 message digest. That adversary could then
        forge a signature by finding a message with the same MD2
        message digest as one that a signer previously signed, and
        presenting the previous signature as the signature on the
        new message. This attack would succeed only if the signer
        previously used MD2, since the DigestInfo value contains
        the message-digest algorithm. If a signer never trusted
        the MD2 algorithm and always used MD5, then the compromise
        of MD2 would not affect the signer. If the DigestInfo value
        contained only the message digest, however, the compromise
        of MD2 would affect signers that use any message-digest
        algorithm.

   4.   There is potential for ambiguity due to the fact
        that the DigestInfo value does not indicate whether the
        digest field contains just the message digest of the
        content or the message digest of the complete DER encoding
        of the authenticatedAttributes field. In other words, it is
        possible for an adversary to transform a signature on
        authenticated attributes to one that appears to be just on
        content by changing the content to be the DER encoding of
        the authenticatedAttributes field, and then removing the
        authenticatedAttributes field. (The reverse transformation
        is possible, but requires that the content be the DER
        encoding of an authenticated attributes value, which is
        unlikely.) This ambiguity is not a new problem, nor is it a
        significant one, as context will generally prevent misuse.
        Indeed, it is also possible for an adversary to transform a
        signature on a certificate or certificate-revocation list
        to one that appears to be just on signed-data content.

9.5 Compatibility with Privacy-Enhanced Mail

   Compatibility with the MIC-ONLY and MIC-CLEAR process types in PEM
   occurs when the content type of the ContentInfo value being signed is
   data, there are no authenticated attributes, the message-digest
   algorithm is md2 or md5, and the digest-encryption algorithm is PKCS
   #1's rsaEncryption. Under all those conditions, the encrypted message
   digest produced here matches the one produced in PEM because:

   1.   The value input to the message-digest algorithm in
        PEM is the same as in this document when there are no
        authenticated attributes. MD2 and MD5 in PEM are the same
        as md2 and md5.

2.   The value encrypted with the signer's private key
     in PEM (as specified in RFC 1423) is the same as in this
     document when there are no authenticated attributes. RSA
     private-key encryption in PEM is the same as PKCS #1's
     rsaEncryption.

The other parts of the signed-data content type (certificates, CRLs,
algorithm identifiers, etc.) are easily translated to and from their
corresponding PEM components.

10. Enveloped-data content type

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

It is expected that the typical application of the enveloped-data
content type will be to represent one or more recipients' digital
envelopes on content of the data, digested-data, or signed-data
content types.

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

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

2.   For each recipient, the content-encryption key is
     encrypted with the recipient's public key.

3.   For each recipient, the encrypted content-
     encryption key and other recipient-specific information are
     collected into a RecipientInfo value, defined in Section
     10.2.

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 10.3 for discussion.)

5.   The RecipientInfo values for all the recipients
     are collected together with the encrypted content into a
     EnvelopedData value, defined in Section 10.1.

A recipient opens the envelope by decrypting the one of the encrypted content-encryption keys with the recipient's private key and decrypting the encrypted content with the recovered content-encryption key. The recipient's private key is referenced by an issuer distinguished name and an issuer-specific serial number that uniquely identify the certificate for the corresponding public 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.

This content type is not compatible with Privacy-Enhanced Mail (although some processes it defines are compatible with their PEM counterparts), since Privacy-Enhanced Mail always involves digital signatures, never digital envelopes alone.

10.1 EnvelopedData type

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

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

RecipientInfos ::= SET OF RecipientInfo

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

EncryptedContent ::= OCTET STRING
```

The fields of type EnvelopedData have the following meanings:

> o   version is the syntax version number. It shall be
>   0 for this version of the document.

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

> o   encryptedContentInfo is the encrypted content
>   information.

The fields of type EncryptedContentInfo have the following meanings:

> o   contentType indicates the type of content.
>
> o   contentEncryptionAlgorithm identifies the content-
>   encryption algorithm (and any associated
>   parameters) under which the content is encrypted.
>   The content-encryption process is described in
>   Section 10.3. This algorithm is the same for all
>   recipients.
>
> o   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.

Note. The fact that the recipientInfos field comes before the
encryptedContentInfo field makes it possible to process an
EnvelopedData value in a single pass. (Single-pass processing is
described in Section 5.)

10.2 RecipientInfo type

Per-recipient information is represented in the type RecipientInfo:

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

    KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey }

EncryptedKey ::= OCTET STRING
```

The fields of type RecipientInfo have the following meanings:

> o   version is the syntax version number. It shall be
>   0 for this version of the document.
>
> o   issuerAndSerialNumber specifies the recipient's
>   certificate (and thereby the recipient's
>   distinguished name and public key) by issuer
>   distinguished name and issuer-specific serial
>   number.

        o   keyEncryptionAlgorithm identifies the key-
                encryption algorithm (and any associated
                parameters) under which the content-encryption key
                is encrypted with the recipient's public key. The
                key-encryption process is described in Section
                10.4.

        o   encryptedKey is the result of encrypting the
                content-encryption key with the recipient's public
                key (see below).

10.3 Content-encryption process

   The input to the content-encryption process is the "value" of the
   content being enveloped. Specifically, the input is the contents
   octets of a definite-length BER encoding of the content field of the
   ContentInfo value to which the enveloping process is applied. Only
   the contents octets of the BER encoding are encrypted, not the
   identifier octets or length octets; those other octets are not
   represented at all.

   When the content being enveloped has content type data, 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. This method is
   compatible with Privacy-Enhanced Mail.

   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-and-enveloped-data content type be
   employed, or that the digested-data and enveloped-data content types
   be applied in succession.

   Notes.

        1.   The reason that a definite-length BER encoding is
                required is that the bit indicating whether the length is
                definite or indefinite is not recorded anywhere in the
                enveloped-data content type. Definite-length encoding is
                more appropriate for simple types such as octet strings, so
                definite-length encoding is chosen.

2.   Some content-encryption algorithms assume the
      input length is a multiple of k octets, where k > 1, and
      let the application define a method for handling inputs
      whose lengths are not a multiple of k octets. For such
      algorithms, the method shall be to pad the input 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 and no padding string is a suffix of another. This
      padding method is well-defined if and only if k < 256;
      methods for larger k are an open issue for further study.

10.4 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.

11. Signed-and-enveloped-data content type

   This section defines the signed-and-enveloped-data content type. For
   brevity, much of this section is expressed in terms of material in
   Sections 9 and 10.

   The signed-and-enveloped-data content type consists of encrypted
   content of any type, encrypted content-encryption keys for one or
   more recipients, and doubly encrypted message digests for one or more
   signers. The "double encryption" consists of an encryption with a
   signer's private key followed by an encryption with the content-
   encryption key.

   The combination of encrypted content and encrypted content-encryption
   key for a recipient is a "digital envelope" for that recipient. The
   recovered singly encrypted message digest for a signer is a "digital
   signature" on the recovered content for that signer. Any type of
   content can be enveloped for any number of recipients and signed by
   any number of signers in parallel.

It is expected that the typical application of the signed-and-
enveloped-data content type will be to represent one signer's digital
signature and one or more recipients' digital envelopes on content of
the data content type.

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

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

    2.   For each recipient, the content-encryption key is
         encrypted with the recipient's public key.

    3.   For each recipient, the encrypted content-
         encryption key and other recipient-specific
         information are collected into a RecipientInfo
         value, defined in Section 10.2.

    4.   For each signer, a message digest 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.)

    5.   For each signer, the message digest and associated
         information are encrypted with the signer's
         private key, and the result is encrypted with the
         content-encryption key. (The second encryption may
         require that the result of the first encryption be
         padded to a multiple of some block size; see
         Section 10.3 for discussion.)

    6.   For each signer, the doubly encrypted message
         digest and other signer-specific information are
         collected into a SignerInfo value, defined in
         Section 9.2.

    7.   The content is encrypted with the content-
         encryption key. (See Section 10.3 for discussion.)

    8.   The message-digest algorithms for all the signers,
         the SignerInfo values for all the signers and the
         RecipientInfo values for all the recipients are
         collected together with the encrypted content into
         a SignedAndEnvelopedData value, defined in Section
         11.1.

A recipient opens the envelope and verifies the signatures in two
steps. First, the one of the encrypted content-encryption keys is
decrypted with the recipient's private key, and the encrypted content
is decrypted with the recovered content-encryption key. Second, the
doubly encrypted message digest for each signer is decrypted with the
recovered content-encryption key, the result is decrypted with the
signer's public key, and the recovered message digest is compared to
an independently computed message digest.

Recipient private keys and signer public keys are contained or
referenced as discussed in Sections 9 and 10.

This section is divided into three parts. The first part describes
the top-level type SignedAndEnvelopedData and the second part
describes the digest-encryption process. Other types and processes
are the same as in Sections 9 and 10. The third part summarizes
compatibility with Privacy-Enhanced Mail.

Note. The signed-and-enveloped-data content type provides
cryptographic enhancements similar to those resulting from the
sequential combination of signed-data and enveloped-data content
types. However, since the signed-and-enveloped-data content type does
not have authenticated or unauthenticated attributes, nor does it
provide enveloping of signer information other than the signature,
the sequential combination of signed-data and enveloped-data content
types is generally preferable to the SignedAndEnvelopedData content
type, except when compatibility with the ENCRYPTED process type in
Privacy-Enhanced Mail in intended.

11.1 SignedAndEnvelopedData type

   The signed-and-enveloped-data content type shall have ASN.1 type
   SignedAndEnvelopedData:

   SignedAndEnvelopedData ::= SEQUENCE {
     version Version,
     recipientInfos RecipientInfos,
     digestAlgorithms DigestAlgorithmIdentifiers,
     encryptedContentInfo EncryptedContentInfo,
     certificates
        [0] IMPLICIT ExtendedCertificatesAndCertificates
          OPTIONAL,
     crls
       [1] IMPLICIT CertificateRevocationLists OPTIONAL,
     signerInfos SignerInfos }

The fields of type SignedAndEnvelopedData have the following
meanings:

    o   version is the syntax version number. It shall be
            1 for this version of the document.

    o   recipientInfos is a collection of per-recipient
            information, as in Section 10. There must be at
            least one element in the collection.

    o   digestAlgorithms is a collection of message-digest
            algorithm identifiers, as in Section 9. The
            message-digesting process is the same as in
            Section 9 in the case when there are no
            authenticated attributes.

    o   encryptedContentInfo is the encrypted content, as
            in Section 10. It can have any of the defined
            content types.

    o   certificates is a set of PKCS #6 extended
            certificates and X.509 certificates, as in Section
            9.

    o   crls is a set of certificate-revocation lists, as
            in Section 9.

    o   signerInfos is a collection of per-signer
            information. There must be at least one element in
            the collection. SignerInfo values have the same
            meaning as in Section 9 with the exception of the
            encryptedDigest field (see below).

    Notes.

    1.   The fact that the recipientInfos and
            digestAlgorithms fields come before the contentInfo field
            and the signerInfos field comes after it makes it possible
            to process a SignedAndEnvelopedData value in a single pass.
            (Single-pass processing is described in Section 5.)

    2.   The difference between version 1
            SignedAndEnvelopedData and version 0 SignedAndEnvelopedData
            (defined in PKCS #7, Version 1.4) is that the crls field is
            allowed in version 1, but not in version 0. Except for the
            difference in version number, version 0
            SignedAndEnvelopedData values are acceptable as version 1
            values. An implementation can therefore process

SignedAndEnvelopedData values of either version as though
they were version 1 values. It is suggested that PKCS
implementations generate only version 1
SignedAndEnvelopedData values, but be prepared to process
SignedAndEnvelopedData values of either version.

11.2 Digest-encryption process

The input to the digest-encryption process is the same as in Section
9, but the process itself is different. Specifically, the process
involves two steps. First, the input to the process is supplied to
the signer's digest-encryption algorithm, as in Section 9. Second,
the result of the first step is encrypted with the content-encryption
key. There is no DER encoding between the two steps; the "value"
output by the first step is input directly to the second step. (See
Section 10.3 for discussion.)

This process is compatible with the ENCRYPTED process type in
Privacy-Enhanced Mail.

Note. The purpose of the second step is to prevent an adversary from
recovering the message digest of the content. Otherwise, an
adversary would be able to determine which of a list of candidate
contents (e.g., "Yes" or "No") is the actual content by comparing the
their message digests to the actual message digest.

11.3 Compatibility with Privacy-Enhanced Mail

Compatibility with the ENCRYPTED process type of PEM occurs when the
content type of the ContentInfo value being signed and enveloped is
data, the message-digest algorithm is md2 or md5, the content-
encryption algorithm is DES in CBC mode, the digest-encryption
algorithm is PKCS #1's rsaEncryption, and the key-encryption
algorithm is PKCS #1's rsaEncryption. Under all those conditions,
the doubly encrypted message digest and the encrypted content
encryption key match the ones produced in PEM because of reasons
similar to those given in Section 9.5, as well as the following:

    1.   The value input to the content-encryption
         algorithm in PEM is the same as in this document.
         DES in CBC mode is the same as desCBC.

    2.   The value input to the key-encryption algorithm in
         PEM is the same as in this document (see Section
         10.4). RSA public-key encryption in PEM is the
         same as PKCS #1's rsaEncryption.

        3.   The double-encryption process applied to the
              message digest in this document and in PEM are the
              same.

   The other parts of the signed-and-enveloped-data content type
   (certificates, CRLs, algorithm identifiers, etc.) are easily
   translated to and from their corresponding PEM components. (CRLs are
   carried in a separate PEM message.)

12. Digested-data content type

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

   It is expected that the typical application of the digested-data
   content type will be to add integrity to content of the data content
   type, and that the result would become the content input to the
   enveloped-data content type.

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

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

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

   Digest ::= OCTET STRING

   The fields of type DigestedData have the following meanings:

        o   version is the syntax version number. It shall be
              0 for this version of the document.

        o   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 9 in the case when there are no
             authenticated attributes.)

        o   contentInfo is the content that is digested. It
             can have any of the defined content types.

        o   digest is the result of the message-digesting process.

   Note. The fact that the digestAlgorithm field comes before the
   contentInfo field and the digest field comes after it makes it
   possible to process a DigestedData value in a single pass. (Single-
   pass processing is described in Section 5.)

13. 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 are assumed to be managed by other means.

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

   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:

        o   version is the syntax version number. It shall be
             0 for this version of the document.

        o   encryptedContentInfo is the encrypted content
             information, as in Section 10.

14. Object identifiers

   This document defines seven object identifiers: pkcs-7, data,
   signedData, envelopedData, signedAndEnvelopedData, digestedData, and
   encryptedData.

   The object identifier pkcs-7 identifies this document.

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

   The object identifiers data, signedData, envelopedData,
   signedAndEnvelopedData, digestedData, and encryptedData, identify,
   respectively, the data, signed-data, enveloped-data, signed-and-
   enveloped-data, digested-data, and encrypted-data content types
   defined in Sections 8-13.

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

   These object identifiers are intended to be used in the contentType
   field of a value of type ContentInfo (see Section 5). The content
   field of that type, which has the content-type-specific syntax ANY
   DEFINED BY contentType, would have ASN.1 type Data, SignedData,
   EnvelopedData, SignedAndEnvelopedData, DigestedData, and
   EncryptedData, respectively. These object identifiers are also
   intended to be used in a PKCS #9 content-type attribute.

Security Considerations

   Security issues are discussed throughout this memo.

Revision history

   Versions 1.0-1.3

   Versions 1.0-1.3 were distributed to participants in RSA Data
   Security, Inc.'s Public-Key Cryptography Standards meetings in
   February and March 1991.

   Version 1.4

   Version 1.4 is part of the June 3, 1991 initial public release of
   PKCS. Version 1.4 was published as NIST/OSI Implementors' Workshop
   document SEC-SIG-91-22.

Version 1.5

Version 1.5 incorporates several editorial changes, including updates
to the references and the addition of a revision history. The
following substantive changes were made:

    o   Section 6: CertificateRevocationLists type is
         added.

    o   Section 9.1: SignedData syntax is revised. The new
         version allows for the dissemination of
         certificate-revocation lists along with
         signatures. It also allows for the dissemination
         of certificates and certificate-revocation lists
         alone, without any signatures.

    o   Section 9.2: SignerInfo syntax is revised. The new
         version includes a message-digest encryption
         process compatible with Privacy-Enhanced Mail as
         specified in RFC 1423.

    o   Section 9.3: Meaning of "the DER encoding of the
         authenticatedAttributes field" is clarified as
         "the DER encoding of the Attributes value."

    o   Section 10.3: Padding method for content-
         encryption algorithms is described.

    o   Section 11.1: SignedAndEnvelopedData syntax is
         revised. The new version allows for the
         dissemination of certificate-revocation lists.

    o   Section 13: Encrypted-data content type is added.
         This content type consists of encrypted content of
         any type.

    o   Section 14: encryptedData object identifier is
         added.

Supersedes June 3, 1991 version, which was also published as NIST/OSI
Implementors' Workshop document SEC-SIG-91-22.

Acknowledgements

Author's Address

   Burt Kaliski
   RSA Laboratories East
   20 Crosby Drive
   Bedford, MA 01730

   Phone: (617) 687-7000
   EMail: burt@rsa.com

Full Copyright Statement