S/MIME Version 3 Message Specification

Status of this Memo

Copyright Notice

1. Introduction

   S/MIME (Secure/Multipurpose Internet Mail Extensions) provides a
   consistent way to send and receive secure MIME data. Based on the
   popular Internet MIME standard, S/MIME provides the following
   cryptographic security services for electronic messaging
   applications: authentication, message integrity and non-repudiation
   of origin (using digital signatures) and privacy and data security
   (using encryption).

   S/MIME can be used by traditional mail user agents (MUAs) to add
   cryptographic security services to mail that is sent, and to
   interpret cryptographic security services in mail that is received.
   However, S/MIME is not restricted to mail; it can be used with any
   transport mechanism that transports MIME data, such as HTTP. As such,
   S/MIME takes advantage of the object-based features of MIME and
   allows secure messages to be exchanged in mixed-transport systems.

   Further, S/MIME can be used in automated message transfer agents that
   use cryptographic security services that do not require any human
   intervention, such as the signing of software-generated documents and
   the encryption of FAX messages sent over the Internet.

1.1 Specification Overview

   This document describes a protocol for adding cryptographic signature
   and encryption services to MIME data. The MIME standard [MIME-SPEC]
   provides a general structure for the content type of Internet
   messages and allows extensions for new content type applications.

This memo defines how to create a MIME body part that has been
cryptographically enhanced according to CMS [CMS], which is derived
from PKCS #7 [PKCS-7]. This memo also defines the application/pkcs7-
mime MIME type that can be used to transport those body parts.

This memo also discusses how to use the multipart/signed MIME type
defined in [MIME-SECURE] to transport S/MIME signed messages. This
memo also defines the application/pkcs7-signature MIME type, which is
also used to transport S/MIME signed messages.

In order to create S/MIME messages, an S/MIME agent has to follow
specifications in this memo, as well as the specifications listed in
the Cryptographic Message Syntax [CMS].

Throughout this memo, there are requirements and recommendations made
for how receiving agents handle incoming messages. There are separate
requirements and recommendations for how sending agents create
outgoing messages. In general, the best strategy is to "be liberal in
what you receive and conservative in what you send". Most of the
requirements are placed on the handling of incoming messages while
the recommendations are mostly on the creation of outgoing messages.

The separation for requirements on receiving agents and sending
agents also derives from the likelihood that there will be S/MIME
systems that involve software other than traditional Internet mail
clients. S/MIME can be used with any system that transports MIME
data. An automated process that sends an encrypted message might not
be able to receive an encrypted message at all, for example. Thus,
the requirements and recommendations for the two types of agents are
listed separately when appropriate.

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [MUSTSHOULD].

1.3 Definitions

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

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

BER: Basic Encoding Rules for ASN.1, as defined in CCITT X.209.

Certificate: A type that binds an entity's distinguished name to a
public key with a digital signature.

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

7-bit data: Text data with lines less than 998 characters long, where
none of the characters have the 8th bit set, and there are no NULL
characters. <CR> and <LF> occur only as part of a <CR><LF> end of
line delimiter.

8-bit data: Text data with lines less than 998 characters, and where
none of the characters are NULL characters. <CR> and <LF> occur only
as part of a <CR><LF> end of line delimiter.

Binary data: Arbitrary data.

Transfer Encoding: A reversible transformation made on data so 8-bit
or binary data may be sent via a channel that only transmits 7-bit
data.

Receiving agent: software that interprets and processes S/MIME CMS
objects, MIME body parts that contain CMS objects, or both.

Sending agent: software that creates S/MIME CMS objects, MIME body
parts that contain CMS objects, or both.

S/MIME agent: user software that is a receiving agent, a sending
agent, or both.

1.4 Compatibility with Prior Practice of S/MIME

S/MIME version 3 agents should attempt to have the greatest
interoperability possible with S/MIME version 2 agents. S/MIME
version 2 is described in RFC 2311 through RFC 2315, inclusive. RFC
2311 also has historical information about the development of S/MIME.

2. CMS Options

CMS allows for a wide variety of options in content and algorithm
support. This section puts forth a number of support requirements and
recommendations in order to achieve a base level of interoperability
among all S/MIME implementations. [CMS] provides additional details
regarding the use of the cryptographic algorithms.

2.1 DigestAlgorithmIdentifier

Sending and receiving agents MUST support SHA-1 [SHA1]. Receiving
agents SHOULD support MD5 [MD5] for the purpose of providing backward
compatibility with MD5-digested S/MIME v2 SignedData objects.

2.2 SignatureAlgorithmIdentifier

   Sending and receiving agents MUST support id-dsa defined in [DSS].
   The algorithm parameters MUST be absent (not encoded as NULL).

   Receiving agents SHOULD support rsaEncryption, defined in [PKCS-1].

   Sending agents SHOULD support rsaEncryption. Outgoing messages are
   signed with a user's private key. The size of the private key is
   determined during key generation.

   Note that S/MIME v2 clients are only capable of verifying digital
   signatures using the rsaEncryption algorithm.

2.3 KeyEncryptionAlgorithmIdentifier

   Sending and receiving agents MUST support Diffie-Hellman defined in
   [DH].

   Receiving agents SHOULD support rsaEncryption. Incoming encrypted
   messages contain symmetric keys which are to be decrypted with a
   user's private key. The size of the private key is determined during
   key generation.

   Sending agents SHOULD support rsaEncryption.

   Note that S/MIME v2 clients are only capable of decrypting content
   encryption keys using the rsaEncryption algorithm.

2.4 General Syntax

   CMS defines multiple content types. Of these, only the Data,
   SignedData, and EnvelopedData content types are currently used for
   S/MIME.

2.4.1 Data Content Type

   Sending agents MUST use the id-data content type identifier to
   indicate the message content which has had security services applied
   to it. For example, when applying a digital signature to MIME data,
   the CMS signedData encapContentInfo eContentType MUST include the
   id-data object identifier and the MIME content MUST be stored in the
   SignedData encapContentInfo eContent OCTET STRING (unless the sending
   agent is using multipart/signed, in which case the eContent is
   absent, per section 3.4.3 of this document). As another example,
   when applying encryption to MIME data, the CMS EnvelopedData

encryptedContentInfo ContentType MUST include the id-data object
identifier and the encrypted MIME content MUST be stored in the
envelopedData encryptedContentInfo encryptedContent OCTET STRING.

2.4.2 SignedData Content Type

Sending agents MUST use the signedData content type to apply a
digital signature to a message or, in a degenerate case where there
is no signature information, to convey certificates.

2.4.3 EnvelopedData Content Type

This content type is used to apply privacy protection to a message. A
sender needs to have access to a public key for each intended message
recipient to use this service. This content type does not provide
authentication.

2.5 Attribute SignerInfo Type

The SignerInfo type allows the inclusion of unsigned and signed
attributes to be included along with a signature.

Receiving agents MUST be able to handle zero or one instance of each
of the signed attributes listed here. Sending agents SHOULD generate
one instance of each of the following signed attributes in each
S/MIME message:

- signingTime (section 2.5.1 in this document)
- sMIMECapabilities (section 2.5.2 in this document)
- sMIMEEncryptionKeyPreference (section 2.5.3 in this document)

Further, receiving agents SHOULD be able to handle zero or one
instance in the signed attributes of the signingCertificate attribute
(section 5 in [ESS]).

Sending agents SHOULD generate one instance of the signingCertificate
signed attribute in each S/MIME message.

Additional attributes and values for these attributes may be defined
in the future. Receiving agents SHOULD handle attributes or values
that it does not recognize in a graceful manner.

Sending agents that include signed attributes that are not listed
here SHOULD display those attributes to the user, so that the user is
aware of all of the data being signed.

2.5.1 Signing-Time Attribute

   The signing-time attribute is used to convey the time that a message
   was signed. Until there are trusted timestamping services, the time
   of signing will most likely be created by a message originator and
   therefore is only as trustworthy as the originator.

   Sending agents MUST encode signing time through the year 2049 as
   UTCTime; signing times in 2050 or later MUST be encoded as
   GeneralizedTime. When the UTCTime CHOICE is used, S/MIME agents MUST
   interpret the year field (YY) as follows:

   if YY is greater than or equal to 50, the year is interpreted as
   19YY; if YY is less than 50, the year is interpreted as 20YY.

2.5.2 SMIMECapabilities Attribute

   The SMIMECapabilities attribute includes signature algorithms (such
   as "sha1WithRSAEncryption"), symmetric algorithms (such as "DES-
   EDE3-CBC"), and key encipherment algorithms (such as
   "rsaEncryption"). It also includes a non-algorithm capability which
   is the preference for signedData. The SMIMECapabilities were designed
   to be flexible and extensible so that, in the future, a means of
   identifying other capabilities and preferences such as certificates
   can be added in a way that will not cause current clients to break.

   If present, the SMIMECapabilities attribute MUST be a
   SignedAttribute; it MUST NOT be an UnsignedAttribute. CMS defines
   SignedAttributes as a SET OF Attribute. The SignedAttributes in a
   signerInfo MUST NOT include multiple instances of the
   SMIMECapabilities attribute. CMS defines the ASN.1 syntax for
   Attribute to include attrValues SET OF AttributeValue. A
   SMIMECapabilities attribute MUST only include a single instance of
   AttributeValue. There MUST NOT be zero or multiple instances of
   AttributeValue present in the attrValues SET OF AttributeValue.

   The semantics of the SMIMECapabilites attribute specify a partial
   list as to what the client announcing the SMIMECapabilites can
   support. A client does not have to list every capability it supports,
   and probably should not list all its capabilities so that the
   capabilities list doesn't get too long. In an SMIMECapabilities
   attribute, the OIDs are listed in order of their preference, but
   SHOULD be logically separated along the lines of their categories
   (signature algorithms, symmetric algorithms, key encipherment
   algorithms, etc.)

The structure of the SMIMECapabilities attribute is to facilitate
simple table lookups and binary comparisons in order to determine
matches. For instance, the DER-encoding for the SMIMECapability for
DES EDE3 CBC MUST be identically encoded regardless of the
implementation.

In the case of symmetric algorithms, the associated parameters for
the OID MUST specify all of the parameters necessary to differentiate
between two instances of the same algorithm. For instance, the number
of rounds and block size for RC5 must be specified in addition to the
key length.

There is a list of OIDs (OIDs Used with S/MIME) that is centrally
maintained and is separate from this memo. The list of OIDs is
maintained by the Internet Mail Consortium at
<http://www.imc.org/ietf-smime/oids.html>. Note that all OIDs
associated with the MUST and SHOULD implement algorithms are included
in section A of this document.

The OIDs that correspond to algorithms SHOULD use the same OID as the
actual algorithm, except in the case where the algorithm usage is
ambiguous from the OID. For instance, in an earlier draft,
rsaEncryption was ambiguous because it could refer to either a
signature algorithm or a key encipherment algorithm. In the event
that an OID is ambiguous, it needs to be arbitrated by the maintainer
of the registered SMIMECapabilities list as to which type of
algorithm will use the OID, and a new OID MUST be allocated under the
smimeCapabilities OID to satisfy the other use of the OID.

The registered SMIMECapabilities list specifies the parameters for
OIDs that need them, most notably key lengths in the case of
variable-length symmetric ciphers. In the event that there are no
differentiating parameters for a particular OID, the parameters MUST
be omitted, and MUST NOT be encoded as NULL.

Additional values for the SMIMECapabilities attribute may be defined
in the future. Receiving agents MUST handle a SMIMECapabilities
object that has values that it does not recognize in a graceful
manner.

2.5.3 Encryption Key Preference Attribute

The encryption key preference attribute allows the signer to
unambiguously describe which of the signer's certificates has the
signer's preferred encryption key. This attribute is designed to
enhance behavior for interoperating with those clients which use
separate keys for encryption and signing. This attribute is used to

convey to anyone viewing the attribute which of the listed
certificates should be used for encrypting a session key for future
encrypted messages.

If present, the SMIMEEncryptionKeyPreference attribute MUST be a
SignedAttribute; it MUST NOT be an UnsignedAttribute. CMS defines
SignedAttributes as a SET OF Attribute. The SignedAttributes in a
signerInfo MUST NOT include multiple instances of the
SMIMEEncryptionKeyPreference attribute. CMS defines the ASN.1 syntax
for Attribute to include attrValues SET OF AttributeValue. A
SMIMEEncryptionKeyPreference attribute MUST only include a single
instance of AttributeValue. There MUST NOT be zero or multiple
instances of AttributeValue present in the attrValues SET OF
AttributeValue.

The sending agent SHOULD include the referenced certificate in the
set of certificates included in the signed message if this attribute
is used. The certificate may be omitted if it has been previously
made available to the receiving agent. Sending agents SHOULD use
this attribute if the commonly used or preferred encryption
certificate is not the same as the certificate used to sign the
message.

Receiving agents SHOULD store the preference data if the signature on
the message is valid and the signing time is greater than the
currently stored value. (As with the SMIMECapabilities, the clock
skew should be checked and the data not used if the skew is too
great.) Receiving agents SHOULD respect the sender's encryption key
preference attribute if possible. This however represents only a
preference and the receiving agent may use any certificate in
replying to the sender that is valid.

2.5.3.1 Selection of Recipient Key Management Certificate

   In order to determine the key management certificate to be used when
   sending a future CMS envelopedData message for a particular
   recipient, the following steps SHOULD be followed:

    - If an SMIMEEncryptionKeyPreference attribute is found in a
   signedData object received from the desired recipient, this
   identifies the X.509 certificate that should be used as the X.509
   key management certificate for the recipient.

    - If an SMIMEEncryptionKeyPreference attribute is not found in a
   signedData object received from the desired recipient, the set of
   X.509 certificates should be searched for a X.509 certificate with
   the same subject name as the signing X.509 certificate which can
   be used for key management.

- Or use some other method of determining the user's key management
key. If a X.509 key management certificate is not found, then
encryption cannot be done with the signer of the message. If multiple
X.509 key management certificates are found, the S/MIME agent can
make an arbitrary choice between them.

2.6 SignerIdentifier SignerInfo Type

S/MIME v3 requires the use of SignerInfo version 1, that is the
issuerAndSerialNumber CHOICE MUST be used for SignerIdentifier.

2.7 ContentEncryptionAlgorithmIdentifier

Sending and receiving agents MUST support encryption and decryption
with DES EDE3 CBC, hereinafter called "tripleDES" [3DES] [DES].
Receiving agents SHOULD support encryption and decryption using the
RC2 [RC2] or a compatible algorithm at a key size of 40 bits,
hereinafter called "RC2/40".

2.7.1 Deciding Which Encryption Method To Use

When a sending agent creates an encrypted message, it has to decide
which type of encryption to use. The decision process involves using
information garnered from the capabilities lists included in messages
received from the recipient, as well as out-of-band information such
as private agreements, user preferences, legal restrictions, and so
on.

Section 2.5 defines a method by which a sending agent can optionally
announce, among other things, its decrypting capabilities in its
order of preference. The following method for processing and
remembering the encryption capabilities attribute in incoming signed
messages SHOULD be used.

- If the receiving agent has not yet created a list of capabilities
  for the sender's public key, then, after verifying the signature
  on the incoming message and checking the timestamp, the receiving
  agent SHOULD create a new list containing at least the signing
  time and the symmetric capabilities.

 - If such a list already exists, the receiving agent SHOULD verify
   that the signing time in the incoming message is greater than
   the signing time stored in the list and that the signature is
   valid. If so, the receiving agent SHOULD update both the signing
   time and capabilities in the list. Values of the signing time that
   lie far in the future (that is, a greater discrepancy than any
   reasonable clock skew), or a capabilities list in messages whose
   signature could not be verified, MUST NOT be accepted.

The list of capabilities SHOULD be stored for future use in creating messages.

Before sending a message, the sending agent MUST decide whether it is willing to use weak encryption for the particular data in the message. If the sending agent decides that weak encryption is unacceptable for this data, then the sending agent MUST NOT use a weak algorithm such as RC2/40. The decision to use or not use weak encryption overrides any other decision in this section about which encryption algorithm to use.

Sections 2.7.2.1 through 2.7.2.4 describe the decisions a sending agent SHOULD use in deciding which type of encryption should be applied to a message. These rules are ordered, so the sending agent SHOULD make its decision in the order given.

2.7.1.1 Rule 1: Known Capabilities

If the sending agent has received a set of capabilities from the recipient for the message the agent is about to encrypt, then the sending agent SHOULD use that information by selecting the first capability in the list (that is, the capability most preferred by the intended recipient) for which the sending agent knows how to encrypt. The sending agent SHOULD use one of the capabilities in the list if the agent reasonably expects the recipient to be able to decrypt the message.

2.7.1.2 Rule 2: Unknown Capabilities, Known Use of Encryption

If:
 - the sending agent has no knowledge of the encryption capabilities
   of the recipient,
 - and the sending agent has received at least one message from the
   recipient,
 - and the last encrypted message received from the recipient had a
   trusted signature on it,

then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from the recipient.

2.7.1.3 Rule 3: Unknown Capabilities, Unknown Version of S/MIME

   If:

     - the sending agent has no knowledge of the encryption capabilities
       of the recipient,
     - and the sending agent has no knowledge of the version of S/MIME
       of the recipient,

   then the sending agent SHOULD use tripleDES because it is a stronger
   algorithm and is required by S/MIME v3. If the sending agent chooses
   not to use tripleDES in this step, it SHOULD use RC2/40.

2.7.2 Choosing Weak Encryption

   Like all algorithms that use 40 bit keys, RC2/40 is considered by
   many to be weak encryption. A sending agent that is controlled by a
   human SHOULD allow a human sender to determine the risks of sending
   data using RC2/40 or a similarly weak encryption algorithm before
   sending the data, and possibly allow the human to use a stronger
   encryption method such as tripleDES.

2.7.3 Multiple Recipients

   If a sending agent is composing an encrypted message to a group of
   recipients where the encryption capabilities of some of the
   recipients do not overlap, the sending agent is forced to send more
   than one message. It should be noted that if the sending agent
   chooses to send a message encrypted with a strong algorithm, and then
   send the same message encrypted with a weak algorithm, someone
   watching the communications channel may be able to learn the contents
   of the strongly-encrypted message simply by decrypting the weakly-
   encrypted message.

3. Creating S/MIME Messages

   This section describes the S/MIME message formats and how they are
   created. S/MIME messages are a combination of MIME bodies and CMS
   objects. Several MIME types as well as several CMS objects are used.
   The data to be secured is always a canonical MIME entity. The MIME
   entity and other data, such as certificates and algorithm
   identifiers, are given to CMS processing facilities which produces a
   CMS object. The CMS object is then finally wrapped in MIME. The
   Enhanced Security Services for S/MIME [ESS] document provides
   examples of how nested, secured S/MIME messages are formatted. ESS
   provides an example of how a triple-wrapped S/MIME message is
   formatted using multipart/signed and application/pkcs7-mime for the
   signatures.

S/MIME provides one format for enveloped-only data, several formats
for signed-only data, and several formats for signed and enveloped
data. Several formats are required to accommodate several
environments, in particular for signed messages. The criteria for
choosing among these formats are also described.

The reader of this section is expected to understand MIME as
described in [MIME-SPEC] and [MIME-SECURE].

3.1 Preparing the MIME Entity for Signing or Enveloping

S/MIME is used to secure MIME entities. A MIME entity may be a sub-
part, sub-parts of a message, or the whole message with all its sub-
parts. A MIME entity that is the whole message includes only the MIME
headers and MIME body, and does not include the RFC-822 headers.
Note that S/MIME can also be used to secure MIME entities used in
applications other than Internet mail.

The MIME entity that is secured and described in this section can be
thought of as the "inside" MIME entity. That is, it is the
"innermost" object in what is possibly a larger MIME message.
Processing "outside" MIME entities into CMS objects is described in
Section 3.2, 3.4 and elsewhere.

The procedure for preparing a MIME entity is given in [MIME-SPEC].
The same procedure is used here with some additional restrictions
when signing. Description of the procedures from [MIME-SPEC] are
repeated here, but the reader should refer to that document for the
exact procedure. This section also describes additional requirements.

A single procedure is used for creating MIME entities that are to be
signed, enveloped, or both signed and enveloped. Some additional
steps are recommended to defend against known corruptions that can
occur during mail transport that are of particular importance for
clear- signing using the multipart/signed format. It is recommended
that these additional steps be performed on enveloped messages, or
signed and enveloped messages in order that the message can be
forwarded to any environment without modification.

These steps are descriptive rather than prescriptive. The implementor
is free to use any procedure as long as the result is the same.

Step 1. The MIME entity is prepared according to the local
conventions

Step 2. The leaf parts of the MIME entity are converted to canonical
form

Step 3. Appropriate transfer encoding is applied to the leaves of the
MIME entity

When an S/MIME message is received, the security services on the
message are processed, and the result is the MIME entity. That MIME
entity is typically passed to a MIME-capable user agent where, it is
further decoded and presented to the user or receiving application.

3.1.1 Canonicalization

Each MIME entity MUST be converted to a canonical form that is
uniquely and unambiguously representable in the environment where the
signature is created and the environment where the signature will be
verified. MIME entities MUST be canonicalized for enveloping as well
as signing.

The exact details of canonicalization depend on the actual MIME type
and subtype of an entity, and are not described here. Instead, the
standard for the particular MIME type should be consulted. For
example, canonicalization of type text/plain is different from
canonicalization of audio/basic. Other than text types, most types
have only one representation regardless of computing platform or
environment which can be considered their canonical representation.
In general, canonicalization will be performed by the non-security
part of the sending agent rather than the S/MIME implementation.

The most common and important canonicalization is for text, which is
often represented differently in different environments. MIME
entities of major type "text" must have both their line endings and
character set canonicalized. The line ending must be the pair of
characters <CR><LF>, and the charset should be a registered charset
[CHARSETS]. The details of the canonicalization are specified in
[MIME-SPEC]. The chosen charset SHOULD be named in the charset
parameter so that the receiving agent can unambiguously determine the
charset used.

Note that some charsets such as ISO-2022 have multiple
representations for the same characters. When preparing such text for
signing, the canonical representation specified for the charset MUST
be used.

3.1.2 Transfer Encoding

When generating any of the secured MIME entities below, except the
signing using the multipart/signed format, no transfer encoding at
all is required. S/MIME implementations MUST be able to deal with
binary MIME objects. If no Content-Transfer-Encoding header is
present, the transfer encoding should be considered 7BIT.

S/MIME implementations SHOULD however use transfer encoding described
in section 3.1.3 for all MIME entities they secure. The reason for
securing only 7-bit MIME entities, even for enveloped data that are
not exposed to the transport, is that it allows the MIME entity to be
handled in any environment without changing it. For example, a
trusted gateway might remove the envelope, but not the signature, of
a message, and then forward the signed message on to the end
recipient so that they can verify the signatures directly. If the
transport internal to the site is not 8-bit clean, such as on a
wide-area network with a single mail gateway, verifying the signature
will not be possible unless the original MIME entity was only 7-bit
data.

3.1.3 Transfer Encoding for Signing Using multipart/signed

If a multipart/signed entity is EVER to be transmitted over the
standard Internet SMTP infrastructure or other transport that is
constrained to 7-bit text, it MUST have transfer encoding applied so
that it is represented as 7-bit text. MIME entities that are 7-bit
data already need no transfer encoding. Entities such as 8-bit text
and binary data can be encoded with quoted-printable or base-64
transfer encoding.

The primary reason for the 7-bit requirement is that the Internet
mail transport infrastructure cannot guarantee transport of 8-bit or
binary data. Even though many segments of the transport
infrastructure now handle 8-bit and even binary data, it is sometimes
not possible to know whether the transport path is 8-bit clear. If a
mail message with 8-bit data were to encounter a message transfer
agent that can not transmit 8-bit or binary data, the agent has three
options, none of which are acceptable for a clear-signed message:

- The agent could change the transfer encoding; this would invalidate
  the signature.
- The agent could transmit the data anyway, which would most likely
  result in the 8th bit being corrupted; this too would invalidate the
  signature.
- The agent could return the message to the sender.

[MIME-SECURE] prohibits an agent from changing the transfer encoding
of the first part of a multipart/signed message. If a compliant agent
that can not transmit 8-bit or binary data encounters a
multipart/signed message with 8-bit or binary data in the first part,
it would have to return the message to the sender as undeliverable.

3.1.4 Sample Canonical MIME Entity

   This example shows a multipart/mixed message with full transfer
   encoding. This message contains a text part and an attachment. The
   sample message text includes characters that are not US-ASCII and
   thus must be transfer encoded. Though not shown here, the end of each
   line is <CR><LF>. The line ending of the MIME headers, the text, and
   transfer encoded parts, all must be <CR><LF>.

   Note that this example is not of an S/MIME message.

      Content-Type: multipart/mixed; boundary=bar

      --bar
      Content-Type: text/plain; charset=iso-8859-1
      Content-Transfer-Encoding: quoted-printable

      =A1Hola Michael!

      How do you like the new S/MIME specification?

      I agree. It's generally a good idea to encode lines that begin with
      From=20 because some mail transport agents will insert a
      greater-than (>) sign, thus invalidating the signature.

      Also, in some cases it might be desirable to encode any =20
      trailing whitespace that occurs on lines in order to ensure =20
      that the message signature is not invalidated when passing =20
      a gateway that modifies such whitespace (like BITNET). =20

      --bar
      Content-Type: image/jpeg
      Content-Transfer-Encoding: base64

      iQCVAwUBMJrRF2N9oWBghPDJAQE9UQQAtl7LuRVndBjrk4EqYBIb3h5QXIX/LC//
      jJV5bNvkZIGPIcEmI5iFd9boEgvpirHtIREEqLQRkYNoBActFBZmh9GC3C041WGq
      uMbrbxc+nIs1TIKlA08rVi9ig/2Yh7LFrK5Ein57U/W72vgSxLhe/zhdfolT9Brn
      HOxEa44b+EI=

      --bar--

3.2 The application/pkcs7-mime Type

   The application/pkcs7-mime type is used to carry CMS objects of
   several types including envelopedData and signedData. The details of
   constructing these entities is described in subsequent sections. This
   section describes the general characteristics of the
   application/pkcs7-mime type.

The carried CMS object always contains a MIME entity that is prepared
as described in section 3.1 if the eContentType is id-data. Other
contents may be carried when the eContentType contains different
values. See [ESS] for an example of this with signed receipts.

Since CMS objects are binary data, in most cases base-64 transfer
encoding is appropriate, in particular when used with SMTP transport.
The transfer encoding used depends on the transport through which the
object is to be sent, and is not a characteristic of the MIME type.

Note that this discussion refers to the transfer encoding of the CMS
object or "outside" MIME entity. It is completely distinct from, and
unrelated to, the transfer encoding of the MIME entity secured by the
CMS object, the "inside" object, which is described in section 3.1.

Because there are several types of application/pkcs7-mime objects, a
sending agent SHOULD do as much as possible to help a receiving agent
know about the contents of the object without forcing the receiving
agent to decode the ASN.1 for the object. The MIME headers of all
application/pkcs7-mime objects SHOULD include the optional "smime-
type" parameter, as described in the following sections.

3.2.1 The name and filename Parameters

For the application/pkcs7-mime, sending agents SHOULD emit the
optional "name" parameter to the Content-Type field for compatibility
with older systems. Sending agents SHOULD also emit the optional
Content-Disposition field [CONTDISP] with the "filename" parameter.
If a sending agent emits the above parameters, the value of the
parameters SHOULD be a file name with the appropriate extension:

MIME Type	File Extension
Application/pkcs7-mime (signedData, envelopedData)	.p7m
Application/pkcs7-mime (degenerate signedData "certs-only" message)	.p7c
Application/pkcs7-signature	.p7s

In addition, the file name SHOULD be limited to eight characters
followed by a three letter extension. The eight character filename
base can be any distinct name; the use of the filename base "smime"
SHOULD be used to indicate that the MIME entity is associated with
S/MIME.

Including a file name serves two purposes. It facilitates easier use
of S/MIME objects as files on disk. It also can convey type
information across gateways. When a MIME entity of type
application/pkcs7-mime (for example) arrives at a gateway that has no
special knowledge of S/MIME, it will default the entity's MIME type
to application/octet-stream and treat it as a generic attachment,
thus losing the type information. However, the suggested filename for
an attachment is often carried across a gateway. This often allows
the receiving systems to determine the appropriate application to
hand the attachment off to, in this case a stand-alone S/MIME
processing application. Note that this mechanism is provided as a
convenience for implementations in certain environments. A proper
S/MIME implementation MUST use the MIME types and MUST NOT rely on
the file extensions.

3.2.2 The smime-type parameter

The application/pkcs7-mime content type defines the optional "smime-
type" parameter. The intent of this parameter is to convey details
about the security applied (signed or enveloped) along with
infomation about the contained content. This memo defines the
following smime-types.

Name	Security	Inner Content
enveloped-data	EnvelopedData	id-data
signed-data	SignedData	id-data
certs-only	SignedData	none

In order that consistency can be obtained with future, the following
guidelines should be followed when assigning a new smime-type
parameter.

1. If both signing and encryption can be applied to the content, then
two values for smime-type SHOULD be assigned "signed-*" and
"encrypted-*". If one operation can be assigned then this may be
omitted. Thus since "certs-only" can only be signed, "signed-" is
omitted.

2. A common string for a content oid should be assigned. We use
"data" for the id-data content OID when MIME is the inner content.

3. If no common string is assigned. Then the common string of
"OID.<oid>" is recommended (for example, "OID.1.3.6.1.5.5.7.6.1"
would be DES40).

3.3 Creating an Enveloped-only Message

   This section describes the format for enveloping a MIME entity
   without signing it. It is important to note that sending enveloped
   but not signed messages does not provide for data integrity. It is
   possible to replace ciphertext in such a way that the processed
   message will still be valid, but the meaning may be altered.

   Step 1. The MIME entity to be enveloped is prepared according to
   section 3.1.

   Step 2. The MIME entity and other required data is processed into a
   CMS object of type envelopedData. In addition to encrypting a copy of
   the content-encryption key for each recipient, a copy of the content
   encryption key SHOULD be encrypted for the originator and included in
   the envelopedData (see CMS Section 6).

   Step 3. The CMS object is inserted into an application/pkcs7-mime
   MIME entity.

   The smime-type parameter for enveloped-only messages is "enveloped-
   data". The file extension for this type of message is ".p7m".

   A sample message would be:

      Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
         name=smime.p7m
      Content-Transfer-Encoding: base64
      Content-Disposition: attachment; filename=smime.p7m

      rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
      7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
      f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
      0GhIGfHfQbnj756YT64V

3.4 Creating a Signed-only Message

   There are two formats for signed messages defined for S/MIME:
   application/pkcs7-mime with SignedData, and multipart/signed. In
   general, the multipart/signed form is preferred for sending, and
   receiving agents SHOULD be able to handle both.

3.4.1 Choosing a Format for Signed-only Messages

   There are no hard-and-fast rules when a particular signed-only format
   should be chosen because it depends on the capabilities of all the
   receivers and the relative importance of receivers with S/MIME
   facilities being able to verify the signature versus the importance
   of receivers without S/MIME software being able to view the message.

   Messages signed using the multipart/signed format can always be
   viewed by the receiver whether they have S/MIME software or not. They
   can also be viewed whether they are using a MIME-native user agent or
   they have messages translated by a gateway. In this context, "be
   viewed" means the ability to process the message essentially as if it
   were not a signed message, including any other MIME structure the
   message might have.

   Messages signed using the signedData format cannot be viewed by a
   recipient unless they have S/MIME facilities. However, if they have
   S/MIME facilities, these messages can always be verified if they were
   not changed in transit.

3.4.2 Signing Using application/pkcs7-mime with SignedData

   This signing format uses the application/pkcs7-mime MIME type. The
   steps to create this format are:

   Step 1. The MIME entity is prepared according to section 3.1

   Step 2. The MIME entity and other required data is processed into a
   CMS object of type signedData

   Step 3. The CMS object is inserted into an application/pkcs7-mime
   MIME entity

   The smime-type parameter for messages using application/pkcs7-mime
   with SignedData is "signed-data". The file extension for this type of
   message is ".p7m".

   A sample message would be:

       Content-Type: application/pkcs7-mime; smime-type=signed-data;
           name=smime.p7m
       Content-Transfer-Encoding: base64
       Content-Disposition: attachment; filename=smime.p7m

```
567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH
HUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyHh
6YT64V0GhIGfHfQbnj75
```

3.4.3 Signing Using the multipart/signed Format

This format is a clear-signing format. Recipients without any S/MIME
or CMS processing facilities are able to view the message. It makes
use of the multipart/signed MIME type described in [MIME-SECURE]. The
multipart/signed MIME type has two parts. The first part contains the
MIME entity that is signed; the second part contains the "detached
signature" CMS SignedData object in which the encapContentInfo
eContent field is absent.

3.4.3.1 The application/pkcs7-signature MIME Type

This MIME type always contains a single CMS object of type
signedData. The signedData encapContentInfo eContent field MUST be
absent. The signerInfos field contains the signatures for the MIME
entity.

The file extension for signed-only messages using application/pkcs7-
signature is ".p7s".

3.4.3.2 Creating a multipart/signed Message

Step 1. The MIME entity to be signed is prepared according to section
3.1, taking special care for clear-signing.

Step 2. The MIME entity is presented to CMS processing in order to
obtain an object of type signedData in which the encapContentInfo
eContent field is absent.

Step 3. The MIME entity is inserted into the first part of a
multipart/signed message with no processing other than that described
in section 3.1.

Step 4. Transfer encoding is applied to the "detached signature" CMS
SignedData object and it is inserted into a MIME entity of type
application/pkcs7-signature.

Step 5. The MIME entity of the application/pkcs7-signature is
inserted into the second part of the multipart/signed entity.

The multipart/signed Content type has two required parameters: the
protocol parameter and the micalg parameter.

The protocol parameter MUST be "application/pkcs7-signature". Note
that quotation marks are required around the protocol parameter
because MIME requires that the "/" character in the parameter value
MUST be quoted.

The micalg parameter allows for one-pass processing when the
signature is being verified. The value of the micalg parameter is
dependent on the message digest algorithm(s) used in the calculation
of the Message Integrity Check. If multiple message digest algorithms
are used they MUST be separated by commas per [MIME-SECURE]. The
values to be placed in the micalg parameter SHOULD be from the
following:

```
Algorithm    Value
used

MD5          md5
SHA-1        sha1
Any other    unknown
```

(Historical note: some early implementations of S/MIME emitted and
expected "rsa-md5" and "rsa-sha1" for the micalg parameter.)
Receiving agents SHOULD be able to recover gracefully from a micalg
parameter value that they do not recognize.

3.4.3.3 Sample multipart/signed Message

```
Content-Type: multipart/signed;
   protocol="application/pkcs7-signature";
   micalg=sha1; boundary=boundary42

--boundary42
Content-Type: text/plain

This is a clear-signed message.

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

3.5 Signing and Encrypting

   To achieve signing and enveloping, any of the signed-only and
   encrypted-only formats may be nested. This is allowed because the
   above formats are all MIME entities, and because they all secure MIME
   entities.

   An S/MIME implementation MUST be able to receive and process
   arbitrarily nested S/MIME within reasonable resource limits of the
   recipient computer.

   It is possible to either sign a message first, or to envelope the
   message first. It is up to the implementor and the user to choose.
   When signing first, the signatories are then securely obscured by the
   enveloping. When enveloping first the signatories are exposed, but it
   is possible to verify signatures without removing the enveloping.
   This may be useful in an environment were automatic signature
   verification is desired, as no private key material is required to
   verify a signature.

   There are security ramifications to choosing whether to sign first or
   encrypt first. A recipient of a message that is encrypted and then
   signed can validate that the encrypted block was unaltered, but
   cannot determine any relationship between the signer and the
   unencrypted contents of the message. A recipient of a message that is
   signed-then-encrypted can assume that the signed message itself has
   not been altered, but that a careful attacker may have changed the
   unauthenticated portions of the encrypted message.

3.6 Creating a Certificates-only Message

   The certificates only message or MIME entity is used to transport
   certificates, such as in response to a registration request. This
   format can also be used to convey CRLs.

   Step 1. The certificates are made available to the CMS generating
   process which creates a CMS object of type signedData. The signedData
   encapContentInfo eContent field MUST be absent and signerInfos field
   MUST be empty.

   Step 2. The CMS signedData object is enclosed in an
   application/pkcs7-mime MIME entity

   The smime-type parameter for a certs-only message is "certs-only".
   The file extension for this type of message is ".p7c".

3.7 Registration Requests

   A sending agent that signs messages MUST have a certificate for the
   signature so that a receiving agent can verify the signature. There
   are many ways of getting certificates, such as through an exchange
   with a certificate authority, through a hardware token or diskette,
   and so on.

   S/MIME v2 [SMIMEV2] specified a method for "registering" public keys
   with certificate authorities using an application/pkcs10 body part.
   The IETF's PKIX Working Group is preparing another method for
   requesting certificates; however, that work was not finished at the
   time of this memo. S/MIME v3 does not specify how to request a

   certificate, but instead mandates that every sending agent already
   has a certificate. Standardization of certificate management is being
   pursued separately in the IETF.

3.8 Identifying an S/MIME Message

   Because S/MIME takes into account interoperation in non-MIME
   environments, several different mechanisms are employed to carry the
   type information, and it becomes a bit difficult to identify S/MIME
   messages. The following table lists criteria for determining whether
   or not a message is an S/MIME message. A message is considered an
   S/MIME message if it matches any below.

   The file suffix in the table below comes from the "name" parameter in
   the content-type header, or the "filename" parameter on the content-
   disposition header. These parameters that give the file suffix are
   not listed below as part of the parameter section.

   MIME type: application/pkcs7-mime
   parameters: any
   file suffix: any

   MIME type: multipart/signed
   parameters: protocol="application/pkcs7-signature"
   file suffix: any

   MIME type: application/octet-stream
   parameters: any
   file suffix: p7m, p7s, p7c

4. Certificate Processing

   A receiving agent MUST provide some certificate retrieval mechanism
   in order to gain access to certificates for recipients of digital
   envelopes. This memo does not cover how S/MIME agents handle
   certificates, only what they do after a certificate has been
   validated or rejected. S/MIME certification issues are covered in
   [CERT3].

   At a minimum, for initial S/MIME deployment, a user agent could
   automatically generate a message to an intended recipient requesting
   that recipient's certificate in a signed return message. Receiving
   and sending agents SHOULD also provide a mechanism to allow a user to
   "store and protect" certificates for correspondents in such a way so
   as to guarantee their later retrieval.

4.1 Key Pair Generation

   If an S/MIME agent needs to generate a key pair, then the S/MIME
   agent or some related administrative utility or function MUST be
   capable of generating separate DH and DSS public/private key pairs on
   behalf of the user. Each key pair MUST be generated from a good
   source of non-deterministic random input [RANDOM] and the private key
   MUST be protected in a secure fashion.

   If an S/MIME agent needs to generate a key pair, then the S/MIME
   agent or some related administrative utility or function SHOULD
   generate RSA key pairs.

   A user agent SHOULD generate RSA key pairs at a minimum key size of
   768 bits. A user agent MUST NOT generate RSA key pairs less than 512
   bits long. Creating keys longer than 1024 bits may cause some older
   S/MIME receiving agents to not be able to verify signatures, but
   gives better security and is therefore valuable. A receiving agent
   SHOULD be able to verify signatures with keys of any size over 512
   bits. Some agents created in the United States have chosen to create
   512 bit keys in order to get more advantageous export licenses.
   However, 512 bit keys are considered by many to be cryptographically
   insecure. Implementors should be aware that multiple (active) key
   pairs may be associated with a single individual. For example, one
   key pair may be used to support confidentiality, while a different
   key pair may be used for authentication.

5. Security

   This entire memo discusses security. Security issues not covered in
   other parts of the memo include:

   40-bit encryption is considered weak by most cryptographers. Using
   weak cryptography in S/MIME offers little actual security over
   sending plaintext. However, other features of S/MIME, such as the
   specification of tripleDES and the ability to announce stronger
   cryptographic capabilities to parties with whom you communicate,
   allow senders to create messages that use strong encryption. Using
   weak cryptography is never recommended unless the only alternative is
   no cryptography. When feasible, sending and receiving agents should
   inform senders and recipients the relative cryptographic strength of
   messages.

   It is impossible for most software or people to estimate the value of
   a message. Further, it is impossible for most software or people to
   estimate the actual cost of decrypting a message that is encrypted
   with a key of a particular size. Further, it is quite difficult to
   determine the cost of a failed decryption if a recipient cannot
   decode a message. Thus, choosing between different key sizes (or
   choosing whether to just use plaintext) is also impossible. However,
   decisions based on these criteria are made all the time, and
   therefore this memo gives a framework for using those estimates in
   choosing algorithms.

   If a sending agent is sending the same message using different
   strengths of cryptography, an attacker watching the communications
   channel may be able to determine the contents of the strongly-
   encrypted message by decrypting the weakly-encrypted version. In
   other words, a sender should not send a copy of a message using
   weaker cryptography than they would use for the original of the
   message.

   Modification of the ciphertext can go undetected if authentication is
   not also used, which is the case when sending EnvelopedData without
   wrapping it in SignedData or enclosing SignedData within it.

A. ASN.1 Module

```
SecureMimeMessageV3
   { iso(1) member-body(2) us(840) rsadsi(113549)
         pkcs(1) pkcs-9(9) smime(16) modules(0) smime(4) }


DEFINITIONS IMPLICIT TAGS ::=
BEGIN


IMPORTS
-- Cryptographic Message Syntax
    SubjectKeyIdentifier, IssuerAndSerialNumber,
RecipientKeyIdentifier
        FROM    CryptographicMessageSyntax
               { iso(1) member-body(2) us(840) rsadsi(113549)
                 pkcs(1) pkcs-9(9) smime(16) modules(0) cms(1) };


--  id-aa is the arc with all new authenticated and unauthenticated
--  attributes produced the by S/MIME Working Group

id-aa OBJECT IDENTIFIER ::= {iso(1) member-body(2) usa(840)
rsadsi(113549)
        pkcs(1) pkcs-9(9) smime(16) attributes(2)}

-- S/MIME Capabilities provides a method of broadcasting the symetric
-- capabilities understood.  Algorithms should be ordered by preference
-- and grouped by type

smimeCapabilities OBJECT IDENTIFIER ::=
   {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 15}


SMIMECapability ::= SEQUENCE {
   capabilityID OBJECT IDENTIFIER,
   parameters ANY DEFINED BY capabilityID OPTIONAL }


SMIMECapabilities ::= SEQUENCE OF SMIMECapability

-- Encryption Key Preference provides a method of broadcasting the
-- preferred encryption certificate.

id-aa-encrypKeyPref OBJECT IDENTIFIER ::= {id-aa 11}

SMIMEEncryptionKeyPreference ::= CHOICE {
   issuerAndSerialNumber   [0] IssuerAndSerialNumber,
   receipentKeyId          [1] RecipientKeyIdentifier,
   subjectAltKeyIdentifier [2] SubjectKeyIdentifier
}
```

-- The Content Encryption Algorithms defined for SMIME are:

-- Triple-DES is the manditory algorithm with CBCParameter being the
-- parameters

dES-EDE3-CBC OBJECT IDENTIFIER ::=
   {iso(1) member-body(2) us(840) rsadsi(113549)
encryptionAlgorithm(3) 7}

CBCParameter ::= IV

IV ::= OCTET STRING (SIZE (8..8))

-- RC2 (or compatable) is an optional algorithm w/ RC2-CBC-paramter
-- as the parameter

rC2-CBC OBJECT IDENTIFIER ::=
   {iso(1) member-body(2) us(840) rsadsi(113549)
encryptionAlgorithm(3) 2}

-- For the effective-key-bits (key size) greater than 32 and less than
-- 256, the RC2-CBC algorithm parameters are encoded as:

RC2-CBC-parameter ::= SEQUENCE {
   rc2ParameterVersion INTEGER,
   iv   IV}

-- For the effective-key-bits of 40, 64, and 128, the
-- rc2ParameterVersion values are 160, 120, 58 respectively.

-- The following list the OIDs to be used with S/MIME V3

-- Digest Algorithms:

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

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

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

```
-- rsa OBJECT IDENTIFIER ::=
--     {joint-iso-ccitt(2) ds(5) algorithm(8) encryptionAlgorithm(1) 1}
--
-- id-dsa OBJECT IDENTIFIER ::=
--     {iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 1 }

-- Signature Algorithms
--
-- md2WithRSAEncryption OBJECT IDENTIFIER ::=
--     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
-- 2}
--
-- md5WithRSAEncryption OBJECT IDENTIFIER ::=
--     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
-- 4}
--
-- sha-1WithRSAEncryption OBJECT IDENTIFIER ::=
--     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
-- 5}
--
-- id-dsa-with-sha1 OBJECT IDENTIFIER ::=
--     {iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 3}

-- Other Signed Attributes
--
-- signingTime OBJECT IDENTIFIER ::=
--     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
-- 5}
--     See [CMS] for a description of how to encode the attribute
--     value.

END
```

B. References

    [3DES]   ANSI X9.52-1998, "Triple Data Encryption Algorithm
                    Modes of Operation", American National Standards
                    Institute, 1998.

    [CERT3]   Ramsdell, B., Editor, "S/MIME Version 3 Certificate
                    Handling", RFC 2632, June 1999.

    [CHARSETS]   Character sets assigned by IANA. See
                    <ftp://ftp.isi.edu/in-
                    notes/iana/assignments/character-sets>.

    [CMS]   Housley, R., "Cryptographic Message Syntax", RFC 2630,
                    June 1999.

    [CONTDISP]   Troost, R., Dorner, S. and K. Moore, "Communicating
                    Presentation Information in Internet Messages: The
                    Content-Disposition Header Field", RFC 2183, August
                    1997.

    [DES]   ANSI X3.106, "American National Standard for
                    Information Systems- Data Link Encryption," American
                    National Standards Institute, 1983.

    [DH]   Rescorla, E., "Diffie-Hellman Key Agreement Method",
                    RFC 2631, June 1999.

    [DSS]   NIST FIPS PUB 186, "Digital Signature Standard", 18
                    May 1994.

    [ESS]   Hoffman, P., Editor "Enhanced Security Services for
                    S/MIME", RFC 2634, June 1999.

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

    [MIME-SPEC]   The primary definition of MIME. "MIME Part 1: Format
                    of Internet Message Bodies", RFC 2045; "MIME Part 2:
                    Media Types", RFC 2046; "MIME Part 3: Message Header
                    Extensions for Non-ASCII Text", RFC 2047; "MIME Part
                    4: Registration Procedures", RFC 2048; "MIME Part 5:
                    Conformance Criteria and Examples", RFC 2049,
                    September 1993.

    [MIME-SECURE]   Galvin, J., Murphy, S., Crocker, S. and N. Freed,
                    "Security Multiparts for MIME: Multipart/Signed and
                    Multipart/Encrypted", RFC 1847, October 1995.

    [MUSTSHOULD]   Bradner, S., "Key words for use in RFCs to Indicate
                     Requirement Levels", BCP14, RFC 2119, March 1997.

    [PKCS-1]   Kaliski, B., "PKCS #1: RSA Encryption Version 2.0",
                     RFC 2437, October 1998.

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

    [RANDOM]   Eastlake, 3rd, D., Crocker, S. and J. Schiller,
                     "Randomness Recommendations for Security", RFC 1750,
                     December 1994.

    [RC2]   Rivest, R., "A Description of the RC2 (r) Encryption
                     Algorithm", RFC 2268, January 1998.

    [SHA1]   NIST FIPS PUB 180-1, "Secure Hash Standard," National
                     Institute of Standards and Technology, U.S. Department
                     of Commerce, DRAFT, 31May 1994.

    [SMIMEV2]   Dusse, S., Hoffman, P., Ramsdell, B., Lundblade, L.
                     and L. Repka, "S/MIME Version 2 Message
                     Specification", RFC 2311, March 1998.

C. Acknowledgements

   Many thanks go out to the other authors of the S/MIME Version 2
   Message Specification RFC: Steve Dusse, Paul Hoffman, Laurence
   Lundblade and Lisa Repka. Without v2, there wouldn't be a v3.

   A number of the members of the S/MIME Working Group have also worked
   very hard and contributed to this document. Any list of people is
   doomed to omission, and for that I apologize. In alphabetical order,
   the following people stand out in my mind due to the fact that they
   made direct contributions to this document.

   Dave Crocker
   Bill Flanigan
   Paul Hoffman
   Russ Housley
   John Pawling
   Jim Schaad

Editor's Address

   Blake Ramsdell
   Worldtalk
   17720 NE 65th St Ste 201
   Redmond, WA 98052

   Phone: +1 425 376 0225
   EMail: blaker@deming.com

Full Copyright Statement

Acknowledgement