S/MIME Version 3 Message Specification

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 draft 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 draft also defines the application/pkcs7-mime MIME type that can be used to transport those body parts.
This draft also discusses how to use the multipart/signed MIME type
defined in [MIME-SECURE] to transport S/MIME signed messages. This
draft 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 agent has to follow
specifications in this draft, as well as the specifications listed in
the Cryptographic Message Syntax [CMS].

Throughout this draft, 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

Throughout this draft, the terms MUST, MUST NOT, SHOULD, and SHOULD
NOT are used in capital letters. This conforms to the definitions in
[MUSTSHOULD]. [MUSTSHOULD] defines the use of these key words to help
make the intent of standards track documents as clear as possible. The
same key words are used in this document to help implementors achieve
interoperability.

1.3 Definitions

For the purposes of this draft, 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.

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.

1.5 Discussion of This Draft

This draft is being discussed on the "ietf-smime" mailing list. To subscribe, send a message to:

ietf-smime-request@imc.org

with the single word

subscribe

in the body of the message. There is a Web site for the mailing list at <http://www.imc.org/ietf-smime/>.

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]. Receiving agents SHOULD support verification of signatures using RSA public key sizes from 512 bits to 1024 bits.

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.

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. If an agent supports rsaEncryption, then it MUST support encryption of symmetric keys with RSA public keys at key sizes from 512 bits to 1024 bits.

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. 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 unauthenticated and authenticated 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 described in this section.

Sending agents SHOULD be able to generate one instance of each of the signed attributes described in this section, and SHOULD include the signing time and SMIMECapabilities attribute in each signed 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 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, 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 "md5WithRSAEncryption"), symmetric algorithms (such as "DES-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.

The semantics of the SMIMECapabilities attribute specify a partial list as to what the client announcing the SMIMECapabilities 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 (the registered SMIMECapabilities list) that is centrally maintained and is separate from this draft. The list of
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 for the signer to unambiguously describe which of the certificates issued to the signer should be used when sending encrypted content. This attribute allows for the signer to state a preference, but not a requirement, as to the certificate to be used. 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 the receiver which of the certificates should be used for encrypting the session key.

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 to great.) Receiving agents SHOULD respect the senders 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 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 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".

2.6.1 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 2.6.2.1 through 2.6.2.4 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.

2.6.1.1 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.

2.6.1.2 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.

2.6.1.3 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.

2.6.2 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.

2.6.3 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 can decipher 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 removed, 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 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
```
Hola 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=20because 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 trailing whitespace that occurs on lines in order to ensure that the message signature is not invalidated when passing a gateway that modifies such whitespace (like BITNET).

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

iQCVAwUBMJrRF2N9oWBghPDJAQE9UQQAt17LuRVndBjrk4EqYBIb3h5QXIX/LC/jjV5bNvk2IGPtcEm151Fd9boEgypirHtIREEqLQRkYNoBActFBZmh9GC3C041WGqumMrbrxc+nIs1TIKlA08rV19ig/2Yh7LFrK5Ein57U/W72vgSxLhe/zhdfolT9BrnHOxEa44b+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:

<table>
<thead>
<tr>
<th>MIME Type</th>
<th>File Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/pkcs7-mime (signedData, envelopedData)</td>
<td>.p7m</td>
</tr>
<tr>
<td>application/pkcs7-mime (degenerate signedData &quot;certs-only&quot; message)</td>
<td>.p7c</td>
</tr>
<tr>
<td>application/pkcs7-signature</td>
<td>.p7s</td>
</tr>
</tbody>
</table>

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.

3.3 Creating an Enveloped-only Message

This section describes the format for enveloping a MIME entity without
signing it.

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
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.

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.

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
```
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 details of the registered type are given in Appendix E.

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:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>md5</td>
</tr>
<tr>
<td>SHA-1</td>
<td>sha1</td>
</tr>
<tr>
<td>Any other</td>
<td>unknown</td>
</tr>
</tbody>
</table>

(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
4VQpfyF467GhIGfHfYT6jH77n8HHHgyHhHUujhJhj756tbB9HGTrfvbnj
n8HHGTrfvJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpFyF4
7GhIGfHfYT64VQbnj756

--boundary42--

3.4.3.4 Encapsulating multipart/signed Messages

Some mail gateways will split or alter a multipart/signed message in ways that might invalidate the signature. Sending agents that create multipart/signed messages may encapsulate those messages using the application/mime construct [APP-MIME], as described in Appendix F.

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 where automatic signature verification is desired, as no private key material is required to verify a signature.

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 draft. 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

A receiving agent MUST provide some certificate retrieval mechanism in order to gain access to certificates for recipients of digital envelopes. This draft 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 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 draft discusses security. Security issues not covered in other parts of the draft 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 draft 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 can 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.

A. Object Identifiers and Syntax

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

SMIMECapabilities ::= SEQUENCE OF SMIMECapability

SMIMEEncryptionKeyPreference ::= CHOICE {
    issuerAndSerialNumber   [0] IssuerAndSerialNumber,
    recipientKeyId          [1] RecipientKeyIdentifier,
    subjectAltKeyIdentifier [2] KeyIdentifier
}

A.1 Content Encryption Algorithms

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 OCTET STRING (8)}

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

DES-ED3-CBC OBJECT IDENTIFIER ::=  
   {iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) 7}

For DES-CBC and DES-ED3-CBC, the parameter should be encoded as:

CBCParameter ::= IV

where IV ::= OCTET STRING -- 8 octets.

A.2 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}

A.3 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}

A.4 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}

A.5 Signed Attributes

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

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


[CONTDISP] "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183


[DH] ANSI X9.42 TBD


[MD5] "The MD5 Message Digest Algorithm", RFC 1321


[MUSTSHOULD] "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119

[PKCS-1] "PKCS #1: RSA Encryption Version 1.5", RFC 2313

[PKCS-7] "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315

[RC2] "A Description of the RC2 (r) Encryption Algorithm", RFC 2268


C. Encapsulating Signed Messages for Internet Transport
The rationale behind the multiple formats for signing has to do with the MIME subtype defaulting rules of the application and multipart top-level types, and the behavior of currently deployed gateways and mail user agents.

Ideally, the multipart/signed format would be the only format used because it provides a truly backwards compatible way to sign MIME entities. In a pure MIME environment with very capable user agents, this would be possible. The world, however, is more complex than this. One problem with the multipart/signed format occurs with gateways to non-MIME environments. In these environments, the gateway will generally not be S/MIME aware, will not recognize the multipart/signed type, and will default its treatment to multipart/mixed as is prescribed by the MIME standard. The real problem occurs when the gateway also applies conversions to the MIME structure of the original message that is being signed and is contained in the first part of the multipart/signed structure, such as the gateway converting text and attachments to the local format. Because the signature is over the MIME structure of the original message, but the original message is now decomposed and transformed, the signature cannot be verified. Because MIME encoding of a particular set of body parts can be done in many different ways, there is no way to reconstruct the original MIME entity over which the signature was computed.

A similar problem occurs when an attempt is made to combine an existing user agent with a stand-alone S/MIME facility. Typical user agents do not have the ability to make a multipart sub-entity available to a stand-alone application in the same way they make leaf MIME entities available to "viewer" applications. This user agent behavior is not required by the MIME standard and thus not widely implemented. The result is that it is impossible for most user agents to hand off the entire multipart/signed entity to a stand-alone application.

C.1 Solutions to the Problem

To work around these two problems, the application/pkcs7-mime type can be used. When going through a gateway, it will be defaulted to the MIME type of application/octet-stream and treated as a single opaque entity. That is, the message will be treated as an attachment of unknown type, converted into the local representation for an attachment and thus can be made available to an S/MIME facility completely intact. A similar result is achieved when a user agent similarly treats the application/pkcs7-mime MIME entity as a simple leaf node of the MIME structure and makes it available to viewer applications.

Another way to work around these problems is to encapsulate the multipart/signed MIME entity in a MIME entity of type application/mime. The result is similar to that obtained using application/pkcs7-mime. When the application/mime entity arrives at a gateway that does not recognize it, its type will be defaulted to application/octet-stream and it will be treated as a single opaque entity. A similar situation will happen with a receiving client that does not recognize the entity. It will usually be treated as a file attachment. It can then be made available to the S/MIME facility.

The major difference between the two alternatives (application/pkcs7-mime or multipart/signed wrapped with application/mime) is when the S/MIME facility opens the attachment. In the latter case, the S/MIME
agent will find a multipart/signed entity rather than a BER encoded CMS object. Considering the two representations abstractly, the only difference is syntax.

Application/mime is a general mechanism for encapsulating MIME, and in particular delaying its interpretation until it can be done in the appropriate environment or at the request of the user. The application/mime specification does not permit a user agent to automatically interpret the encapsulated MIME unless it can be processed entirely and properly. The parameters to the application/mime entity give the type of the encapsulated entity so it can be determined whether or not the entity can be processed before it is expanded.

Application/mime is a general encapsulation mechanism that can be built into a gateway or user agent, allowing expansion of the encapsulated entity under user control. Because it is a general mechanism, it is in many cases more likely to be available than an S/MIME facility. Thus, it enables users to expand or to verify signed messages based on their local facilities and choices. It provides exactly the same advantages that the application/pkcs7-mime with signedData does. It also has the added benefit of allowing expansion in non-S/MIME environments and expansion under the recipient’s control.

C.2 Encapsulation Using application/mime

In some cases, multipart/signed entities are automatically decomposed in such a way as to make computing the hash of the first part, the signed part, impossible; in such a situation, the signature becomes unverifiable. In order to prevent such decomposition until the MIME entity can be processed in a proper S/MIME environment, a multipart/signed entity may be encapsulated in an application/mime entity.

All S/MIME implementations SHOULD be able to generate and receive application/mime encapsulations of multipart/signed entities which have their signature of type application/pkcs7-mime. In particular, on receipt of a MIME entity of type application/mime with the type parameter "multipart/signed" and the protocol parameter "application/pkcs7-mime", a receiving agent SHOULD be able to process the entity correctly. This is required even if the local environment has facilities for processing application/mime because application/mime requires that the encapsulated entity only be processed on request of the user, or if processing software can process the entity completely and correctly. In this case, an S/MIME facility can always process the entity completely and SHOULD do so.

The steps to create an application/mime encapsulation of a multipart/signed entity are:

Step 1. Prepare a multipart/signed message as described in section 3.4.3.2

Step 2. Insert the multipart/signed entity into an application/mime according to [APP-MIME]. This requires that the parameters of the multipart/signed entity be included as parameters on the application/mime entity.

Note that messages using application/mime are subject to the same
encoding rules as message/* and multipart/* types. The encoding of the application/mime part MUST NOT be binary.

In addition, the application/mime entity SHOULD have a name parameter giving a file name ending with ".aps". It SHOULD also have a content-disposition parameter with the same filename. The ".aps" extension SHOULD be used exclusively for application/mime encapsulated multipart/signed entities containing a signature of type application/pkcs7-signature. This is necessary so that the receiving agent can correctly dispatch to software that verifies S/MIME signatures in environments where the MIME type and parameters have been lost or can’t be used for such dispatch. Basically, the file extension becomes the sole carrier of type information.

A sample application/mime encapsulation of a signed message might be:

```
Content-type: application/mime; content-type="multipart/signed";
protocol="application/pkcs7-signature";
micalg=sha1; name=smime.aps
Content-disposition: attachment; filename=smime.aps

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

--boundary42
Content-Type: text/plain

This is a very short clear-signed message. However, at least you can read it!

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s
ghyHhHUujjHhjH77n88HGHTvbnj756tbb9HG4VQpFyF467GhIGfHfYT6
4VQpFyF467GH6FGFyYT6j77n88HGghyHhHUujjHhj756tbb9HGTrfnvbnj
88HGTrnvHhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujjFyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

C.3 Encapsulation in an Non-MIME Environment

While this document primarily addresses the Internet, it is useful to compose and receive S/MIME secured messages in non-MIME environments. This is particularly the case when it is desired that security be implemented end-to-end. Other discussion here addresses the receipt of S/MIME messages in non-MIME environments. Here the composition of multipart/signed entities is addressed.

When a message is to be sent in such an environment, the multipart/signed entity is created as described above. That entity is then treated as an opaque stream of bits and added to the message as an attachment. It must have a file name that ends with ".aps", as this is the sole mechanism for recognizing it as an S/MIME message by the receiving agent.

When this message arrives in a MIME environment, it is likely to have
a MIME type of application/octet-stream, with MIME parameters giving the filename for the attachment. If the intervening gateway has carried the file type, it will end in ".aps" and be recognized as an S/MIME message.

D. Acknowledgements

This document is largely derived from [SMIMEV2] written by Steve Dusse, Paul Hoffman, Blake Ramsdell, Laurence Lundblade, and Lisa Repka.

Significant comments and additions were made by John Pawling and Jim Schaad.

E. Needed changes

4.1 keylengths for RSA need to move to CMS
Should SMIMEEncryptionKeyPreference move to CMS?
2.5.3.1 to determine the "same subject name" should this be a check against the subject DN, or both the DN and the subjectAltName extension?
Should most of appendix A be in CMS?

F. Changes from last draft

Changed "Status of this memo" paragraph to reflect new IETF text (Paul Hoffman)
Removed PKCS #1 from 1.1 (this will be covered in CMS)
Removed 2.6.1.4 and changed 2.6.1.3 (Jim Schaad and Paul Hoffman)
Added 2.5.3.1 (Selection of Recipient Key Management Certificate) (Jim Schaad)
Yanked appendix C (MIME type registration) (Paul Hoffman)
Fixed duplicate sentence in 3.7 (Jim Schaad)
Added language to 2.5 to explain that other attributes should be displayed to the user (Paul Hoffman)

G. Editor’s address

Blake Ramsdell
Worldtalk
13122 NE 20th St., Suite C
Bellevue, WA 98005
(425) 882-8861
blaker@deming.com