Network Working Group
Request for Comments: 3851
Obsoletes: 2633
Category: Standards Track

B. Ramsdell, Editor
Sendmail, Inc.

July 2004

Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification

Status of this Memo

Copyright Notice

Abstract

This document defines Secure/Multipurpose Internet Mail Extensions (S/MIME) version 3.1. S/MIME provides a consistent way to send and receive secure MIME data. Digital signatures provide authentication, message integrity, and non-repudiation with proof of origin. Encryption provides data confidentiality. Compression can be used to reduce data size. This document obsoletes RFC 2633.

Table of Contents

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 data confidentiality (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 specification 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 specification also defines the application/pkcs7-mime MIME type that can be used to transport those body parts.

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

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

Throughout this specification, 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 specification, the following definitions apply.

ASN.1: Abstract Syntax Notation One, as defined in CCITT X.208 [X.208-88].

BER: Basic Encoding Rules for ASN.1, as defined in CCITT X.209 [X.209-88].

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

DER: Distinguished Encoding Rules for ASN.1, as defined in CCITT X.509 [X.509-88].

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 can 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 content types, or both.

Sending agent: Software that creates S/MIME CMS content types, MIME body parts that contain CMS content types, 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 3.1 agents SHOULD attempt to have the greatest interoperability possible with agents for prior versions of S/MIME. S/MIME version 2 is described in RFC 2311 through RFC 2315, inclusive and S/MIME version 3 is described in RFC 2630 through RFC 2634 inclusive. RFC 2311 also has historical information about the development of S/MIME.

1.5. Changes Since S/MIME v3

The RSA public key algorithm was changed to a MUST implement key wrapping algorithm, and the Diffie-Hellman algorithm changed to a SHOULD implement.

The AES symmetric encryption algorithm has been included as a SHOULD implement.

The RSA public key algorithm was changed to a MUST implement signature algorithm.

Ambiguous language about the use of "empty" SignedData messages to transmit certificates was clarified to reflect that transmission of certificate revocation lists is also allowed.

The use of binary encoding for some MIME entities is now explicitly discussed.

Header protection through the use of the message/rfc822 MIME type has been added.

Use of the CompressedData CMS type is allowed, along with required MIME type and file extension additions.

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. [CMSALG] provides additional details regarding the use of the cryptographic algorithms.

2.1. DigestAlgorithmIdentifier

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

2.2. SignatureAlgorithmIdentifier

Receiving agents MUST support id-dsa-with-sha1 defined in [CMSALG]. The algorithm parameters MUST be absent (not encoded as NULL). Receiving agents MUST support rsaEncryption, defined in [CMSALG].

Sending agents MUST support either id-dsa-with-sha1 or rsaEncryption.

If using rsaEncryption, sending and receiving agents MUST support the digest algorithms in section 2.1 as specified.

Note that S/MIME v3 clients might only implement signing or signature verification using id-dsa-with-sha1, and might also use id-dsa as an AlgorithmIdentifier in this field. Receiving clients SHOULD recognize id-dsa as equivalent to id-dsa-with-sha1, and sending clients MUST use id-dsa-with-sha1 if using that algorithm. Also note that S/MIME v2 clients are only required to verify digital signatures using the rsaEncryption algorithm with SHA-1 or MD5, and might not implement id-dsa-with-sha1 or id-dsa at all.

2.3. KeyEncryptionAlgorithmIdentifier

Sending and receiving agents MUST support rsaEncryption, defined in [CMSALG].

Sending and receiving agents SHOULD support Diffie-Hellman defined in [CMSALG], using the ephemeral-static mode.

Note that S/MIME v3 clients might only implement key encryption and decryption using the Diffie-Hellman algorithm. Also note that S/MIME v2 clients are only capable of decrypting content-encryption keys using the rsaEncryption algorithm.

2.4. General Syntax

There are several CMS content types. Of these, only the Data, SignedData, EnvelopedData, and CompressedData 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 identify the "inner" MIME message content. 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. Applying a signature to a message provides authentication, message integrity, and non-repudiation of origin.

2.4.3. EnvelopedData Content Type

This content type is used to apply data confidentiality to a message. A sender needs to have access to a public key for each intended message recipient to use this service.

2.4.4. CompressedData Content Type

This content type is used to apply data compression to a message. This content type does not provide authentication, message integrity, non-repudiation, or data confidentiality, and is only used to reduce message size.

See section 3.6 for further guidance on the use of this type in conjunction with other CMS types.

2.5. Attributes and the 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)
- id-messageDigest (section 11.2 in [CMS])
- id-contentType (section 11.1 in [CMS])

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

Sending agents SHOULD generate one instance of the signingCertificate signed attribute in each SignerInfo structure.

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

Interactive 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. 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"). There are also several identifiers which indicate support for other optional features such as binary encoding and compression. 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 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 need not list all its capabilities so that the capabilities list doesn't get too long. In an SMIMECapabilities attribute, the object identifiers (OIDs) are listed in order of their preference, but SHOULD be separated logically 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. Because of the requirement for identical encoding, individuals documenting algorithms to be used in the SMIMECapabilities attribute SHOULD explicitly document the correct byte sequence for the common cases.

For any capability, 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 the block size for RC5 needs to be specified in addition to the key length.

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 specification, 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 might be defined in the future. Receiving agents MUST handle a SMIMECapabilities object that has values that it does not recognize in a graceful manner.

Section 2.7.1 explains a strategy for caching capabilities.

2.5.2.1. SMIMECapability For the RC2 Algorithm

For the RC2 algorithm preference SMIMECapability, the capabilityID MUST be set to the value rc2-cbc as defined in [CMSALG]. The parameters field MUST contain SMIMECapabilitiesParametersForRC2CBC (see appendix A).

Please note that the SMIMECapabilitiesParametersForRC2CBC is a single INTEGER which contains the effective key length (NOT the corresponding RC2 parameter version value). So, for example, for RC2 with a 128-bit effective key length, the parameter would be encoded as the INTEGER value 128, NOT the corresponding parameter version of 58.

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 that use separate keys for encryption and signing. This attribute is used to convey to anyone viewing the attribute which of the listed certificates is appropriate 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 can use any certificate in replying to the sender that is valid.

Section 2.7.1 explains a strategy for caching preference data.

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 of a 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 v3.1 implementations MUST support both issuerAndSerialNumber as well as subjectKeyIdentifier. Messages that use the subjectKeyIdentifier choice cannot be read by S/MIME v2 clients.

It is important to understand that some certificates use a value for subjectKeyIdentifier that is not suitable for uniquely identifying a certificate. Implementations MUST be prepared for multiple certificates for potentially different entities to have the same value for subjectKeyIdentifier, and MUST be prepared to try each

matching certificate during signature verification before indicating an error condition.

2.7. ContentEncryptionAlgorithmIdentifier

Sending and receiving agents MUST support encryption and decryption with DES EDE3 CBC, hereinafter called "tripleDES" [CMSALG]. Receiving agents SHOULD support encryption and decryption using the RC2 [CMSALG] or a compatible algorithm at a key size of 40 bits, hereinafter called "RC2/40". Sending and receiving agents SHOULD support encryption and decryption with AES [CMSAES] at a key size of 128, 192, and 256 bits.

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 2.5.2 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 will 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) that 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, Unknown Version of S/MIME

If the following two conditions are met:
- 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. Please note 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 could 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 content types. Several MIME types as well as several CMS content types 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 produce a CMS object. Finally, the CMS object is wrapped in MIME. The Enhanced Security Services for S/MIME [ESS] document provides descriptions of how nested, secured S/MIME messages are formatted. ESS provides a description 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, Enveloping or Compressing

S/MIME is used to secure MIME entities. A MIME entity can 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. If protection of the RFC-822 headers is required, the use of the message/rfc822 MIME type is explained later in this section.

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 content types 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 it is suggested that the reader 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 have any combination of signing, enveloping, and compressing applied. 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, so that the message can be forwarded to any environment without modification.

These steps are descriptive rather than prescriptive. The implementer 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.

In order to protect outer, non-content related message headers (for instance, the "Subject", "To", "From" and "CC" fields), the sending client MAY wrap a full MIME message in a message/rfc822 wrapper in order to apply S/MIME security services to these headers. It is up to the receiving client to decide how to present these "inner" headers along with the unprotected "outer" headers.

When an S/MIME message is received, if the top-level protected MIME entity has a Content-Type of message/rfc822, it can be assumed that the intent was to provide header protection. This entity SHOULD be presented as the top-level message, taking into account header merging issues as previously discussed.

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 and compressing 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 is required at all. S/MIME implementations MUST be able to deal with binary MIME objects. If no Content-Transfer-Encoding header is present, the transfer encoding is presumed to be 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.

S/MIME implementations which "know" that all intended recipient(s) are capable of handling inner (all but the outermost) binary MIME objects SHOULD use binary encoding as opposed to a 7-bit-safe transfer encoding for the inner entities. The use of a 7-bit-safe encoding (such as base64) would unnecessarily expand the message size. Implementations MAY "know" that recipient implementations are capable of handling inner binary MIME entities either by interpreting the id-cap-preferBinaryInside sMIMECapabilities attribute, by prior agreement, or by other means.

If one or more intended recipients are unable to handle inner binary MIME objects, or if this capability is unknown for any of the intended recipients, S/MIME implementations SHOULD use transfer encoding described in section 3.1.3 for all MIME entities they secure.

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 clean. 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 need to 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?

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   =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 content types including EnvelopedData, SignedData, and CompressedData. 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 content types 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 certificate management message)	.p7c
application/pkcs7-mime (CompressedData)	.p7z
application/pkcs7-signature (SignedData)	.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 information about the contained content. This specification defines the following smime-types.

Name	CMS type	Inner Content
enveloped-data	EnvelopedData	id-data
signed-data	SignedData	id-data
certs-only	SignedData	none
compressed-data	CompressedData	id-data

In order for consistency to be obtained with future specifications, 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 can 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).

It is explicitly intended that this field be a suitable hint for mail client applications to indicate whether a message is "signed" or "encrypted" without having to tunnel into the CMS payload.

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 can 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 EnvelopedData object is wrapped in a CMS ContentInfo object.

Step 4. The ContentInfo 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 MUST 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 is 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, the SignedData format protects the message content from being changed by benign intermediate agents. Such agents might do line wrapping or content-transfer encoding changes which would break the signature.

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 SignedData object is wrapped in a CMS ContentInfo object.

Step 4. The ContentInfo 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 CMS ContentInfo containing 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 used	Value
MD5	md5
SHA-1	sha1
SHA-256	sha256
SHA-384	sha384
SHA-512	sha512
Any other	(defined separately in algorithm profile or "unknown" if not defined)

(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.

The SHA-256, SHA-384, and SHA-512 algorithms [FIPS180-2] are not currently recommended in S/MIME, and are included here for completeness.

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--
```

The content that is digested (the first part of the multipart/signed) are the bytes:

```
43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 65 78 74 2f 70 6c 61 69
6e 0d 0a 0d 0a 54 68 69 73 20 69 73 20 61 20 63 6c 65 61 72 2d 73 69
67 6e 65 64 20 6d 65 73 73 61 67 65 2e 0d 0a
```

3.5. Creating an Compressed-only Message

This section describes the format for compressing a MIME entity. Please note that versions of S/MIME prior to 3.1 did not specify any use of CompressedData, and will not recognize it. The use of a capability to indicate the ability to receive CompressedData is described in [CMSCOMPR] and is the preferred method for compatibility.

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

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

Step 3. The CompressedData object is wrapped in a CMS ContentInfo object.

Step 4. The ContentInfo object is inserted into an application/pkcs7-mime MIME entity.

The smime-type parameter for compressed-only messages is "compressed-data". The file extension for this type of message is ".p7z".

A sample message would be:

```
Content-Type: application/pkcs7-mime; smime-type=compressed-data;
     name=smime.p7z
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7z

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

3.6. Multiple Operations

The signed-only, encrypted-only, and compressed-only MIME formats can be nested. This works because these formats are all MIME entities that encapsulate other 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 apply any of the signing, encrypting, and compressing operations in any order. It is up to the implementer 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 can 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 could have changed the unauthenticated portions of the encrypted message.

When using compression, keep the following guidelines in mind:

- Compression of binary encoded encrypted data is discouraged, since it will not yield significant compression. Base64 encrypted data could very well benefit, however.
- If a lossy compression algorithm is used with signing, you will need to compress first, then sign.

3.7. Creating a Certificate Management Message

The certificate management message or MIME entity is used to transport certificates and/or certificate revocation lists, such as in response to a registration request.

Step 1. The certificates and/or certificate revocation lists 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 SignedData object is wrapped in a CMS ContentInfo object.

Step 3. The ContentInfo object is enclosed in an application/pkcs7-mime MIME entity.

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

3.8. 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. Since that time, the IETF PKIX Working Group has developed other methods for requesting certificates. However, S/MIME v3.1 does not require a particular certificate request mechanism.

3.9. 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 of the criteria listed 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, p7z
```

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 specification does not cover how S/MIME agents handle certificates, only what they do after a certificate has been validated or rejected. S/MIME certificate issues are covered in [CERT31].

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

All generated key pairs 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 an RSA key pair, then the S/MIME agent or some related administrative utility or function SHOULD generate RSA key pairs using the following guidelines. 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 can 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. Implementers SHOULD be aware that multiple (active) key pairs can be associated with a single individual. For example, one key pair can be used to support confidentiality, while a different key pair can be used for authentication.

5. Security Considerations

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 of 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 specification 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 might 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.

See RFC 3218 [MMA] for more information about thwarting the adaptive chosen ciphertext vulnerability in PKCS #1 Version 1.5 implementations.

In some circumstances the use of the Diffie-Hellman key agreement scheme in a prime order subgroup of a large prime p is vulnerable to certain attacks known as "small-subgroup" attacks. Methods exist, however, to prevent these attacks. These methods are described in RFC 2785 [DHSUB].

A. ASN.1 Module

```
SecureMimeMessageV3dot1
  { iso(1) member-body(2) us(840) rsadsi(113549)
         pkcs(1) pkcs-9(9) smime(16) modules(0) msg-v3dot1(21) }

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-2001(14) };


--  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 symmetric
-- 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
}
```

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

id-cap  OBJECT IDENTIFIER ::= { id-smime 11 }

-- The preferBinaryInside indicates an ability to receive messages
-- with binary encoding inside the CMS wrapper

id-cap-preferBinaryInside  OBJECT IDENTIFIER ::= { id-cap 1 }

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

-- Signature Algorithms Not Found in [CMSALG]
--
-- md2WithRSAEncryption OBJECT IDENTIFIER ::=
--    {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
--     2}
--
-- 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.

SMIMECapabilitiesParametersForRC2CBC ::= INTEGER
--        (RC2 Key Length (number of bits))

END
```

B. References

B.1. Normative References

[CERT31] Ramsdell, B., Ed., "S/MIME Version 3.1 Certificate Handling", RFC 3850, July 2004.

[CHARSETS] Character sets assigned by IANA. See http://www.iana.org/assignments/character-sets

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[CMSAES] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.

[CMSALG] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[CMSCOMPR] Gutmann, P., "Compressed Data Content Type for Cryptographic Message Syntax (CMS)", RFC 3274, June 2002.

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

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

[FIPS180-2] "Secure Hash Signature Standard (SHS)", National Institute of Standards and Technology (NIST). FIPS Publication 180-2.

[MIME-SPEC] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

Freed, N., Klensin, J., and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 2048, November 1996.

Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, November 1996.

[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", BCP 14, RFC 2119, March 1997.

[X.208-88] CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). 1988.

[X.209-88] CCITT. Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). 1988.

[X.509-88] CCITT. Recommendation X.509: The Directory - Authentication Framework. 1988.

B.2. Informative References

[DHSUB] Zuccherato, R., "Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME", RFC 2785, March 2000.

[MMA] Rescorla, E., "Preventing the Million Message Attack on Cryptographic Message Syntax", RFC 3218, January 2002.

[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.

[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.

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.

Tony Capel
Piers Chivers
Dave Crocker
Bill Flanigan
Peter Gutmann
Paul Hoffman
Russ Housley
William Ottaway
John Pawling
Jim Schaad

D. Editor's Address

Blake Ramsdell
Sendmail, Inc.
704 228th Ave NE #775
Sammamish, WA 98074

EMail: blake@sendmail.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.