LAMPS
Internet-Draft
Obsoletes: RFC5751 (if approved)
Intended status: Standards Track
Expires: March 2, 2017

J. Schaad
August Cellars
B. Ramsdell
Brute Squad Labs, Inc.
S. Turner
sn3rd

August 29, 2016

Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.5 Message Specification draft-ietf-lamps-rfc5751-bis-01

Abstract

This document defines Secure/Multipurpose Internet Mail Extensions (S/MIME) version 3.5. 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 5751.

Contributing to this document

The source for this draft is being maintained in GitHub. Suggested changes should be submitted as pull requests at <https://github.com/lamps-wg/smime>. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantial issues need to be discussed on the LAMPS mailing list.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 2, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 4
1.1. Specification Overview 4
1.2. Definitions 5
1.3. Conventions Used in This Document 6
1.4. Compatibility with Prior Practice of S/MIME 7
1.5. Changes from S/MIME v3 to S/MIME v3.1 7
1.6. Changes from S/MIME v3.1 to S/MIME v3.2 7
1.7. Changes since S/MIME v3.2 9
2. CMS Options 9
2.1. DigestAlgorithmIdentifier 9
2.2. SignatureAlgorithmIdentifier 10
2.3. KeyEncryptionAlgorithmIdentifier 10
2.4. General Syntax 11
2.4.1. Data Content Type 11
2.4.2. SignedData Content Type 11
2.4.3. EnvelopedData Content Type 12
2.4.4. AuthEnvelopedData Content Type 12
2.4.5. CompressedData Content Type 12
2.5. Attributes and the SignerInfo Type 12

2.5.1. Signing Time Attribute 13
2.5.2. SMIME Capabilities Attribute 13
2.5.3. Encryption Key Preference Attribute 15
2.6. SignerIdentifier SignerInfo Type 16
2.7. ContentEncryptionAlgorithmIdentifier 16
2.7.1. Deciding Which Encryption Method to Use 17
2.7.2. Choosing Weak Encryption 18
2.7.3. Multiple Recipients 18
3. Creating S/MIME Messages 19
3.1. Preparing the MIME Entity for Signing, Enveloping, or Compressing 19
3.1.1. Canonicalization 20
3.1.2. Transfer Encoding 21
3.1.3. Transfer Encoding for Signing Using multipart/signed 22
3.1.4. Sample Canonical MIME Entity 23
3.2. The application/pkcs7-mime Media Type 23
3.2.1. The name and filename Parameters 24
3.2.2. The smime-type Parameter 25
3.3. Creating an Enveloped-Only Message 26
3.4. Creating an Authenticated Enveloped-Only Message 26
3.5. Creating a Signed-Only Message 27
3.5.1. Choosing a Format for Signed-Only Messages 28
3.5.2. Signing Using application/pkcs7-mime with SignedData 28
3.5.3. Signing Using the multipart/signed Format 29
3.6. Creating a Compressed-Only Message 31
3.7. Multiple Operations 32
3.8. Creating a Certificate Management Message 33
3.9. Registration Requests 33
3.10. Identifying an S/MIME Message 33
4. Certificate Processing 34
4.1. Key Pair Generation 34
4.2. Signature Generation 35
4.3. Signature Verification 35
4.4. Encryption 35
4.5. Decryption 35
5. IANA Considerations 36
5.1. Media Type for application/pkcs7-mime 36
5.2. Media Type for application/pkcs7-signature 37
5.3. Register authEnvelopedData smime-type 38
6. Security Considerations 38
7. References 42
7.1. Normative References 42
7.2. Informative References 45
Appendix A. ASN.1 Module 48
Appendix B. Processing of Historic Mail 50
B.1. DigestAlgorithmIdentifier 51
B.2. Signature Algorithms 51
B.3. ContentEncryptionAlgorithmIdentifier 52

Appendix C. Moving S/MIME v2 Message Specification to Historic Status 53
Appendix D. Acknowledgments 53
Authors' Addresses 53

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). As a supplementary service, S/MIME provides for message compression.

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 or SIP. 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 of Internet messages and allows extensions for new content-type-based applications.

This specification defines how to create a MIME body part that has been cryptographically enhanced according to the Cryptographic Message Syntax (CMS) [CMS], which is derived from PKCS #7 [RFC2315]. This specification also defines the application/pkcs7-mime media type that can be used to transport those body parts.

This document also discusses how to use the multipart/signed media type defined in [RFC1847] to transport S/MIME signed messages. multipart/signed is used in conjunction with the application/pkcs7-signature media 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], [RFC3370], [RFC4056], [RFC3560], and [RFC5754].

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. Definitions

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

ASN.1: Abstract Syntax Notation One, as defined in ITU-T Recommendations X.680, X.681, X.682 and X.683 [ASN.1].

BER: Basic Encoding Rules for ASN.1, as defined in ITU-T Recommendation X.690 [X.690].

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 ITU-T Recommendation X.690 [X.690].

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.

Data Integrity Service: A security service that protects againist unauthorized changes to data by insuring that changes to the data are detectable. [RFC4949]

Data Confidentiality: The property that data is not discolsed to system entities unless they have been authorize to know the data. [RFC4949]

Data Origination: The corroboration that the source of the data received is as claimed. [RFC4949].

1.3. Conventions Used in This Document

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 [RFC2119].

We define some additional terms here:

SHOULD+ This term means the same as SHOULD. However, the authors expect that a requirement marked as SHOULD+ will be promoted at some future time to be a MUST.

SHOULD- This term means the same as SHOULD. However, the authors expect that a requirement marked as SHOULD- will be demoted to a MAY in a future version of this document.

MUST- This term means the same as MUST. However, the authors expect that this requirement will no longer be a MUST in a future document. Although its status will be determined at a later time, it is reasonable to expect that if a future revision of a document alters the status of a MUST-requirement, it will remain at least a SHOULD or a SHOULD-.

1.4. Compatibility with Prior Practice of S/MIME

S/MIME version 3.5 agents ought to 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 [SMIMEv2], S/MIME version 3 is described in RFC 2630 through RFC 2634 inclusive and RFC 5035 [SMIMEv3], S/MIME version 3.1 is described in RFC 3850, RFC 3851, RFC 3852, RFC 2634, and RFC 5035 [SMIMEv3.1], and S/MIME version 3.2 is described in [SMIMEv3.2]. RFC 2311 also has historical information about the development of S/MIME.

1.5. Changes from S/MIME v3 to S/MIME v3.1

The RSA public key algorithm was changed to a MUST implement key wrapping algorithm, and the Diffie-Hellman (DH) 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 media type has been added.

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

1.6. Changes from S/MIME v3.1 to S/MIME v3.2

Editorial changes, e.g., replaced "MIME type" with "media type", content-type with Content-Type.

Moved "Conventions Used in This Document" to Section 1.3. Added definitions for SHOULD+, SHOULD-, and MUST-.

Section 1.1 and Appendix A: Added references to RFCs for RSASSA-PSS, RSAES-OAEP, and SHA2 CMS algorithms. Added CMS Multiple Signers Clarification to CMS reference.

Section 1.2: Updated references to ASN.1 to X.680 and BER and DER to X.690.

Section 1.4: Added references to S/MIME MSG 3.1 RFCs.

Section 2.1 (digest algorithm): SHA-256 added as MUST, SHA-1 and MD5 made SHOULD-.

Section 2.2 (signature algorithms): RSA with SHA-256 added as MUST, and DSA with SHA-256 added as SHOULD+, RSA with SHA-1, DSA with SHA-1, and RSA with MD5 changed to SHOULD-, and RSASSA-PSS with SHA-256 added as SHOULD+. Also added note about what S/MIME v3.1 clients support.

Section 2.3 (key encryption): DH changed to SHOULD-, and RSAES-OAEP added as SHOULD+. Elaborated requirements for key wrap algorithm.

Section 2.5.1: Added requirement that receiving agents MUST support both GeneralizedTime and UTCTime.

Section 2.5.2: Replaced reference "sha1WithRSAEncryption" with "sha256WithRSAEncryption", "DES-3EDE-CBC" with "AES-128 CBC", and deleted the RC5 example.

Section 2.5.2.1: Deleted entire section (discussed deprecated RC2).

Section 2.7, 2.7.1, Appendix A: references to RC2/40 removed.

Section 2.7 (content encryption): AES-128 CBC added as MUST, AES-192 and AES-256 CBC SHOULD+, tripleDES now SHOULD-.

Section 2.7.1: Updated pointers from 2.7.2.1 through 2.7.2.4 to 2.7.1.1 to 2.7.1.2.

Section 3.1.1: Removed text about MIME character sets.

Section 3.2.2 and 3.6: Replaced "encrypted" with "enveloped". Update OID example to use AES-128 CBC oid.

Section 3.4.3.2: Replace micalg parameter for SHA-1 with sha-1.

Section 4: Updated reference to CERT v3.2.

Section 4.1: Updated RSA and DSA key size discussion. Moved last four sentences to security considerations. Updated reference to randomness requirements for security.

Section 5: Added IANA registration templates to update media type registry to point to this document as opposed to RFC 2311.

Section 6: Updated security considerations.

Section 7 : Moved references from Appendix B to this section. Updated references. Added informational references to SMIMEv2, SMIMEv3, and SMIMEv3.1.

Appendix C: Added Appendix B to move S/MIME v2 to Historic status.

1.7. Changes since S/MIME v3.2

- Add the use of AuthEnvelopedData, including defining and registering an smime-type value (Section 2.4.4 and Section 3.4).

- Update the content encryption algorithms (Section 2.7): Add AES-256 GCM , remove AES-192 CBC, mark tripleDES as historic.

2. CMS Options

CMS allows for a wide variety of options in content, attributes, 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. [RFC3370] and [RFC5754] provides additional details regarding the use of the cryptographic algorithms. [ESS] provides additional details regarding the use of additional attributes.

2.1. DigestAlgorithmIdentifier

The algorithms here are used for digesting the body of the message and are not the same as the digest algorithms used as part the signature algorithms. The result of this is placed in the message-digest attribute of the signed attributes. It is RECOMMENDED that the algorithm used for digesting the body of the message be of similar or greater strength than the signature algorithm.

Sending and Receiving agents:

- MUST support SHA-256 [RFC5754]

- MUST support SHA-512.

2.2. SignatureAlgorithmIdentifier

Receiving agents:

- MUST support ECDSA with curve P-256 and SHA-256.

- MUST support EdDSA with curve 25519 using PureEdDSA mode.

- MUST- support RSA with SHA-256.

- SHOULD support RSASSA-PSS with SHA-256.

- MUST NOT support EdDSA using the pre-hash mode.

Sending agents:

- MUST support at least one of the following algorithms: RSASSA-PSS with SHA-256, ECDSA with curve P-256 and SHA-256 or EdDSA with curve 25519 using PureEdDSA mode.

- MUST- support RSA with SHA-256.

- MUST NOT support EdDSA using the pre-hash mode.

Both ECDSA and EdDSA are included in the list of required algorithms for political reasons. NIST is unable to provide the seeds that were used to create their standardized curves, this means that there is a section of the community which believes that there might be a backdoor to these curves. The EdDSA curves were created in response to this feeling. However, there are still significant sections of the industry which need to have NIST approved algorithms. For this reason, both sets of curves are represented in the recieving agent list, but only a requirement for one is in the sending agent list.

See Section 4.1 for information on key size and algorithm references.

2.3. KeyEncryptionAlgorithmIdentifier

Receiving and sending agents:

- MUST support RSA Encryption, as specified in [RFC3370].

- SHOULD+ support RSAES-OAEP, as specified in [RFC3560].

- SHOULD- support DH ephemeral-static mode, as specified in [RFC3370] and [SP800-57].

When DH ephemeral-static is used, a key wrap algorithm is also specified in the KeyEncryptionAlgorithmIdentifier [RFC5652]. The underlying encryption functions for the key wrap and content encryption algorithm ([RFC3370] and [RFC3565]) and the key sizes for the two algorithms MUST be the same (e.g., AES-128 key wrap algorithm with AES-128 content encryption algorithm). As AES-128 CBC is the mandatory-to-implement content encryption algorithm, the AES-128 key wrap algorithm MUST also be supported when DH ephemeral-static is used.

Note that S/MIME v3.1 clients might only implement key encryption and decryption using the rsaEncryption algorithm. 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, AuthEnvelopedData, 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 media type 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 3.5.3 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. AuthEnvelopedData Content Type

This content type is used to apply data confidentiality and message integrity to a message. This content type does not provide authentication or non-repudiation. A sender needs to have access to a public key for each intended message recipient to use this service.

2.4.5. 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 the message's size.

See Section 3.7 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 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:

- Signing Time (Section 2.5.1 in this document)
- SMIME Capabilities (Section 2.5.2 in this document)
- Encryption Key Preference (Section 2.5.3 in this document)
- Message Digest (Section 11.2 in [RFC5652])
- Content Type (Section 11.1 in [RFC5652])

Further, receiving agents SHOULD be able to handle zero or one instance of the signingCertificate and signingCertificatev2 signed attributes, as defined in Section 5 of RFC 2634 [ESS] and Section 3 of RFC 5035 [ESS].

Sending agents SHOULD generate one instance of the signingCertificate or signingCertificatev2 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 they do 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.

Receiving agents MUST be able to process signing-time attributes that are encoded in either UTCTime or GeneralizedTime.

2.5.2. SMIME Capabilities Attribute

The SMIMECapabilities attribute includes signature algorithms (such as "sha256WithRSAEncryption"), symmetric algorithms (such as "AES-128 CBC"), authenticated symmetric algorithms (such as "AES-GCM") and key encipherment algorithms (such as "rsaEncryption"). There are also several identifiers that 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 AES-128 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.

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.

2.5.3. 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 signer of a X.509 certificate that 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 v3.5 implementations MUST support both issuerAndSerialNumber and 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 AES-128 GCM and AES-256 GCM [RFC5084].

- MUST- support encryption and decryption with AES-128 CBC [RFC3565].

- SHOULD+ support encryption and decryption with ChaCha20-Poly1305 [RFC7905].

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. The decision to use or not use weak encryption overrides any other decision in this section about which encryption algorithm to use.

Section 2.7.1.1 and Section 2.7.1.2 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 AES-256 GCM because it is a stronger algorithm and is required by S/MIME v3.5. If the sending agent chooses not to use AES-256 GCM in this step, it SHOULD use AES-128 CBC.

2.7.2. Choosing Weak Encryption

All algorithms that use 112-bit keys are 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 a weak encryption algorithm before sending the data, and possibly allow the human to use a stronger encryption method such as AES GCM or AES CBC.

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 media 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 that 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 [RFC1847].

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 message headers and MIME body, and does not include the RFC-822 header. 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 header is required, the use of the message/rfc822 media 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, Section 3.5, 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. The description of the procedures from [MIME-SPEC] is 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 header fields (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 header fields. It is up to the receiving client to decide how to present this "inner" header along with the unprotected "outer" header.

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 media type and subtype of an entity, and are not described here. Instead, the standard for the particular media 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 that 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].

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 field 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 that "know" that all intended recipients 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 cannot 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.

[RFC1847] prohibits an agent from changing the transfer encoding of the first part of a multipart/signed message. If a compliant agent that cannot 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 the 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 Media Type

The application/pkcs7-mime media type is used to carry CMS content types including EnvelopedData, SignedData, and CompressedData. The details of constructing these entities are described in subsequent sections. This section describes the general characteristics of the application/pkcs7-mime media 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 media 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 Content-Type header field 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 [RFC2138] 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:

Media 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 media 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 media 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	id-data
compressed-data	CompressedData	id-data
authEnvelopedData	AuthEnvelopedData	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 "enveloped-*". 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.2.16.840.1.101.3.4.1.2" would be AES-128 CBC).

It is explicitly intended that this field be a suitable hint for mail client applications to indicate whether a message is "signed" or "enveloped" 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 [RFC5652], 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 an Authenticated Enveloped-Only Message

This section describes the format for enveloping a MIME entity without signing it. Authenticated enveloped messages provide confidentiality and integrity. It is important to note that sending

authenticated enveloped messages does not provide for authentication when using S/MIME. It is possible to replace ciphertext in such a way that the processed message will still be valid, but the meaning can be altered. However this is substantially more difficult than it is for an enveloped-only message as the

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 AuthEnvelopedData. 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 AuthEnvelopedData (see [RFC5083]).

Step 3. The AuthEnvelopedData 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 authenticated enveloped-only messages is "authEnvelopedData". The file extension for this type of message is ".p7m".

A sample message would be:

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

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

3.5. Creating a Signed-Only Message

There are two formats for signed messages defined for S/MIME:

- application/pkcs7-mime with SignedData.

- multipart/signed.

In general, the multipart/signed form is preferred for sending, and receiving agents MUST be able to handle both.

3.5.1. Choosing a Format for Signed-Only Messages

There are no hard-and-fast rules as to when a particular signed-only format is chosen. 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 or not they have S/MIME software. 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 that would break the signature.

3.5.2. Signing Using application/pkcs7-mime with SignedData

This signing format uses the application/pkcs7-mime media 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 are 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
```

3.5.3. 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 media type described in [RFC1847]. The multipart/signed media 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.

3.5.3.1. The application/pkcs7-signature Media Type

This media 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".

3.5.3.2. Creating a multipart/signed Message

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

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

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

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

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

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

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

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

Algorithm	Value Used
MD5	md5
SHA-1	sha-1
SHA-224	sha-224
SHA-256	sha-256
SHA-384	sha-384
SHA-512	sha-512
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", "rsa-sha1", and "sha1" for the micalg parameter.) Receiving agents SHOULD be able to recover gracefully from a micalg parameter value that they do not recognize. Future names for this parameter will be consistent with the IANA "Hash Function Textual Names" registry.

3.5.3.3. Sample multipart/signed Message

```
Content-Type: multipart/signed;
     protocol="application/pkcs7-signature";
     micalg=sha-1; 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) consists of 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
```

3.6. Creating a Compressed-Only Message

This section describes the format for compressing a MIME entity. Please note that versions of S/MIME prior to version 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 [RFC3274] 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 are 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
```

3.7. Multiple Operations

The signed-only, enveloped-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 where 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.

3.8. 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 that 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".

3.9. 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 certification 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 v3.2 does not require a particular certificate request mechanism.

3.10. 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 field, or the "filename" parameter on the Content-Disposition header field. These parameters that give the file suffix are not listed below as part of the parameter section.

Media type	parameters	file suffix
application/pkcs7-mime	any	any
multipart/signed	protocol="application/pkcs7-signature"	any
application/octet-stream	any	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 [RFC5750].

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 [RFC4086] and the private key MUST be protected in a secure fashion.

An S/MIME user agent MUST NOT generate asymmetric keys less than 1024 bits for use with the RSA signature algorithm.

For 512-bit RSA with SHA-1 see [RFC3370] and [FIPS186-2] without Change Notice 1, for 512-bit RSA with SHA-256 see [RFC5754] and [FIPS186-2] without Change Notice 1, and for 1024-bit through 2048-bit RSA with SHA-256 see [RFC5754] and [FIPS186-2] with Change Notice 1. The first reference provides the signature algorithm's object identifier, and the second provides the signature algorithm's definition.

For 512-bit DSA with SHA-1 see [RFC3370] and [FIPS186-2] without Change Notice 1, for 512-bit DSA with SHA-256 see [RFC5754] and [FIPS186-2] without Change Notice 1, for 1024-bit DSA with SHA-1 see [RFC3370] and [FIPS186-2] with Change Notice 1, for 1024-bit and above DSA with SHA-256 see [RFC5754] and [FIPS186-3]. The first reference provides the signature algorithm's object identifier and the second provides the signature algorithm's definition.

For RSASSA-PSS with SHA-256, see [RFC4056]. For 1024-bit DH, see [RFC3370]. For 1024-bit and larger DH, see [SP800-56A]; regardless, use the KDF, which is from X9.42, specified in [RFC3370]. For RSAES-OAEP, see [RFC3560].

4.2. Signature Generation

The following are the requirements for an S/MIME agent generated RSA and RSASSA-PSS signatures:

```
        key size <= 2047 : SHOULD NOT (see Historic Mail Considerations)
2048 <= key size <= 4096 : SHOULD     (see Security Considerations)
4096 <  key size         : MAY        (see Security Considerations)
```

4.3. Signature Verification

The following are the requirements for S/MIME receiving agents during signature verification of RSA and RSASSA-PSS signatures:

```
        key size <= 2047 : SHOULD NOT (see Historic Mail Considerations)
2048 <= key size <= 4096 : MUST       (see Security Considerations)
4096 <  key size         : MAY        (see Security Considerations)
```

4.4. Encryption

The following are the requirements for an S/MIME agent when establishing keys for content encryption using the RSA, RSA-OAEP, and DH algorithms:

```
        key size <= 1023 : SHOULD NOT (see Security Considerations)
1024 <= key size <= 2048 : SHOULD     (see Security Considerations)
2048 <  key size         : MAY        (see Security Considerations)
```

4.5. Decryption

The following are the requirements for an S/MIME agent when establishing keys for content decryption using the RSA, RSAES-OAEP, and DH algorithms:

```
         key size <= 1023 : MAY       (see Security Considerations)
1024 <= key size <= 2048 : MUST      (see Security Considerations)
2048 <  key size         : MAY       (see Security Considerations)
```

5. IANA Considerations

The following information updates the media type registration for application/pkcs7-mime and application/pkcs7-signature to refer to this document as opposed to RFC 2311.

Note that other documents can define additional MIME media types for S/MIME.

5.1. Media Type for application/pkcs7-mime

Type name: application

Subtype Name: pkcs7-mime

Required Parameters: NONE

Optional Parameters: smime-type/signed-data
smime-type/enveloped-data
smime-type/compressed-data
smime-type/certs-only
name

Encoding Considerations: See Section 3 of this document

Security Considerations: See Section 6 of this document

Interoperability Considerations: See Sections 1-6 of this document

Published Specification: RFC 2311, RFC 2633, and this document

Applications that use this media type: Security applications

Additional information: NONE

Person & email to contact for further information:
S/MIME working group chairs smime-chairs@ietf.org

Intended usage: COMMON

Restrictions on usage: NONE

Author: Sean Turner

Change Controller: S/MIME working group delegated from the IESG

5.2. Media Type for application/pkcs7-signature

Type name: application

Subtype Name: pkcs7-signature

Required Parameters: NONE

Optional Parameters: NONE

Encoding Considerations: See Section 3 of this document

Security Considerations: See Section 6 of this document

Interoperability Considerations: See Sections 1-6 of this document

Published Specification: RFC 2311, RFC 2633, and this document

Applications that use this media type: Security applications

Additional information: NONE

Person & email to contact for further information:
S/MIME working group chairs smime-chairs@ietf.org

Intended usage: COMMON

Restrictions on usage: NONE

Author: Sean Turner

Change Controller: S/MIME working group delegated from the IESG

5.3. Register authEnvelopedData smime-type

IANA is required to register the following value in the "Parameter Values for the smime-type Parameter" registry. The values to be registered are:

smime-type value: authEnvelopedData

Reference: [[This Document, Section 3.2.2]]

6. Security Considerations

Cryptographic algorithms will be broken or weakened over time. Implementers and users need to check that the cryptographic algorithms listed in this document continue to provide the expected level of security. The IETF from time to time may issue documents dealing with the current state of the art. For example:

- The Million Message Attack described in RFC 3218 [RFC3218].

- The Diffie-Hellman "small-subgroup" attacks described in RFC 2785 [RFC2785].

- The attacks against hash algorithms described in RFC 4270 [RFC4270].

This specification uses Public-Key Cryptography technologies. It is assumed that the private key is protected to ensure that it is not accessed or altered by unauthorized parties.

It is impossible for most people or software to estimate the value of a message's content. Further, it is impossible for most people or software to estimate the actual cost of recovering an encrypted message content 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 process a message's content. Thus, choosing between different key sizes (or choosing whether to just use plaintext) is also impossible for most people or software. 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.

The choice of 2048 bits as the RSA asymmetric key size in this specification is based on the desire to provide at least 100 bits of security. The key sizes that must be supported to conform to this specification seem appropriate for the Internet based on [RFC3766]. Of course, there are environments, such as financial and medical systems, that may select different key sizes. For this reason, an implementation MAY support key sizes beyond those recommended in this specification.

Receiving agents that validate signatures and sending agents that encrypt messages need to be cautious of cryptographic processing usage when validating signatures and encrypting messages using keys larger than those mandated in this specification. An attacker could send certificates with keys that would result in excessive cryptographic processing, for example, keys larger than those mandated in this specification, which could swamp the processing element. Agents that use such keys without first validating the certificate to a trust anchor are advised to have some sort of cryptographic resource management system to prevent such attacks.

Using weak cryptography in S/MIME offers little actual security over sending plaintext. However, other features of S/MIME, such as the specification of AES 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.

RSA and DSA keys of less than 2048 bits are now considered by many experts to be cryptographically insecure (due to advances in computing power), and should no longer be used to protect messages. Such keys were previously considered secure, so processing previously received signed and encrypted mail will often result in the use of weak keys. Implementations that wish to support previous versions of S/MIME or process old messages need to consider the security risks that result from smaller key sizes (e.g., spoofed messages) versus the costs of denial of service. If an implementation supports verification of digital signatures generated with RSA and DSA keys of less than 1024 bits, it MUST warn the user. Implementers should consider providing different warnings for newly received messages and previously stored messages. Server implementations (e.g., secure mail list servers) where user warnings are not appropriate SHOULD reject messages with weak signatures.

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 digital signatures.

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.

If an implementation is concerned about compliance with National Institute of Standards and Technology (NIST) key size recommendations, then see [SP800-57].

If messaging environments make use of the fact that a message is signed to change the behavior of message processing (examples would be running rules or UI display hints), without first verifying that the message is actually signed and knowing the state of the signature, this can lead to incorrect handling of the message. Visual indicators on messages may need to have the signature

validation code checked periodically if the indicator is supposed to give information on the current status of a message.

Many people assume that the use of an authenticated encryption algorithm is all that is needed to be in a situtation where the sender of the message will be authenticated. In almost all cases this is not a correct statement. There are a number of preconditions that need to hold for an authenticated encryption algorithm to provide this service:

- The starting key must be bound to a single entity. The use of a group key only would allow for the statement that a message was sent by one of the entities that held the key but will not identify a specific entity.

- The message must have exactly one sender and one recipient. Having more than one recipient would allow for the second recipient to create a message that the first recipient would believe is from the sender by stripping them as a recipient from the message.

- A direct path needs to exist from the starting key to the key used as the content encryption key (CEK) which guarantees that no third party could have seen the resulting CEK. This means that one needs to be using an algorithm that is called a "Direct Encryption" or a "Direct Key Agreement" algorithm in other contexts. This means that the starting key is used directly as the CEK key, or that the starting key is used to create a secret which then is transformed into the CEK via a KDF step.

S/MIME implementations almost universally use ephemeral-static rather than static-static key agreement and do not use a pre-existing shared secret when doing encryption, this means that the first precondition is not met. There is a document [RFC6278] which defined how to use static-static key agreement with CMS so that is readably doable. Currently, all S/MIME key agreement methods derive a KEK and wrap a CEK. This violates the third precondition above. New key key agreement algorithms that directly created the CEK without creating an intervening KEK would need to be defined.

Even when all of the preconditions are met and origination of a message is established by the use of an authenticated encryption algorithm, users need to be aware that there is no way to prove this to a third party. This is because either of the parties can successfully create the message (or just alter the content) based on the fact that the CEK is going to be known to both parties. Thus the origination is always built on a presumption that "I did not send this message to myself."

7. References

7.1. Normative References

[ASN.1] "Information Technology - Abstract Syntax Notation (ASN.1)".

ASN.1 syntax consists of the following references [X.680], [X.681], [X.682], and [X.683].

[CHARSETS] "Character sets assigned by IANA.", <http://www.iana.org/assignments/character-sets.>.

[CMS] "Cryptograhic Message Syntax".

This is the set of documents dealing with the cryptographic message syntax and refers to [RFC5652] and [RFC5083].

[ESS] "Enhanced Security Services for S/MIME".

This is the set of documents dealing with enhanged security services and refers to [RFC2634] and [RFC5035].

[FIPS186-2] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS) [With Change Notice 1]", Federal Information Processing Standards Publication 186-2, January 2000.

[FIPS186-3] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication 186-3, June 2009.

[MIME-SPEC] "MIME Message Specifications".

This is the set of documents that define how to use MIME. This set of documents is [RFC2045], [RFC2046], [RFC2047], [RFC2049], [RFC4288], and [RFC4289].

[RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, DOI 10.17487/RFC1847, October 1995, <http://www.rfc-editor.org/info/rfc1847>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <http://www.rfc-editor.org/info/rfc2046>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <http://www.rfc-editor.org/info/rfc2047>.

[RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, DOI 10.17487/RFC2049, November 1996, <http://www.rfc-editor.org/info/rfc2049>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2138] Rigney, C., Rubens, A., Simpson, W., and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2138, DOI 10.17487/RFC2138, April 1997, <http://www.rfc-editor.org/info/rfc2138>.

[RFC2634] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, DOI 10.17487/RFC2634, June 1999, <http://www.rfc-editor.org/info/rfc2634>.

[RFC3274] Gutmann, P., "Compressed Data Content Type for Cryptographic Message Syntax (CMS)", RFC 3274, DOI 10.17487/RFC3274, June 2002, <http://www.rfc-editor.org/info/rfc3274>.

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, DOI 10.17487/RFC3370, August 2002, <http://www.rfc-editor.org/info/rfc3370>.

[RFC3560] Housley, R., "Use of the RSAES-OAEP Key Transport Algorithm in Cryptographic Message Syntax (CMS)", RFC 3560, DOI 10.17487/RFC3560, July 2003, <http://www.rfc-editor.org/info/rfc3560>.

[RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, DOI 10.17487/RFC3565, July 2003, <http://www.rfc-editor.org/info/rfc3565>.

[RFC4056] Schaad, J., "Use of the RSASSA-PSS Signature Algorithm in Cryptographic Message Syntax (CMS)", RFC 4056, DOI 10.17487/RFC4056, June 2005, <http://www.rfc-editor.org/info/rfc4056>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", RFC 4288, DOI 10.17487/RFC4288, December 2005, <http://www.rfc-editor.org/info/rfc4288>.

[RFC4289] Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 4289, DOI 10.17487/RFC4289, December 2005, <http://www.rfc-editor.org/info/rfc4289>.

[RFC5035] Schaad, J., "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", RFC 5035, DOI 10.17487/RFC5035, August 2007, <http://www.rfc-editor.org/info/rfc5035>.

[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, DOI 10.17487/RFC5083, November 2007, <http://www.rfc-editor.org/info/rfc5083>.

[RFC5084] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, DOI 10.17487/RFC5084, November 2007, <http://www.rfc-editor.org/info/rfc5084>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC5754] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, DOI 10.17487/RFC5754, January 2010, <http://www.rfc-editor.org/info/rfc5754>.

[SMIMEv3.5] "S/MIME version 3.5".

This group of documents represents S/MIME version 3.5. This set of documents are [RFC2634], [RFC5750], [[This Document]], [RFC5652], and [RFC5035].

[SP800-56A] National Institute of Standards and Technology (NIST), "Special Publication 800-56A Revision 2: Recommendation Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", May 2013.

[X.680] "Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation. ITU-T Recommendation X.680 (2002)", ITU-T X.680, ISO/IEC 8824-1:2008, November 2008.

[X.681] "Information Technology - Abstract Syntax Notation One (ASN.1): Information object specification", ITU-T X.681, ISO/IEC 8824-2:2008, November 2008.

[X.682] "Information Technology - Abstract Syntax Notation One (ASN.1): Constraint specification", ITU-T X.682, ISO/IEC 8824-3:2008, November 2008.

[X.683] "Information Technology - Abstract Syntax Notation One (ASN.1): Parameteriztion of ASN.1 specifications", ITU-T X.683, ISO/IEC 8824-4:2008, November 2008.

[X.690] "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).", ITU-T X.690, ISO/IEC 8825-1:2002, July 2002.

7.2. Informative References

[RFC2268] Rivest, R., "A Description of the RC2(r) Encryption Algorithm", RFC 2268, DOI 10.17487/RFC2268, March 1998, <http://www.rfc-editor.org/info/rfc2268>.

[RFC2311] Dusse, S., Hoffman, P., Ramsdell, B., Lundblade, L., and L. Repka, "S/MIME Version 2 Message Specification", RFC 2311, DOI 10.17487/RFC2311, March 1998, <http://www.rfc-editor.org/info/rfc2311>.

[RFC2312] Dusse, S., Hoffman, P., Ramsdell, B., and J. Weinstein, "S/MIME Version 2 Certificate Handling", RFC 2312, DOI 10.17487/RFC2312, March 1998, <http://www.rfc-editor.org/info/rfc2312>.

[RFC2313] Kaliski, B., "PKCS #1: RSA Encryption Version 1.5", RFC 2313, DOI 10.17487/RFC2313, March 1998, <http://www.rfc-editor.org/info/rfc2313>.

[RFC2314] Kaliski, B., "PKCS #10: Certification Request Syntax Version 1.5", RFC 2314, DOI 10.17487/RFC2314, March 1998, <http://www.rfc-editor.org/info/rfc2314>.

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <http://www.rfc-editor.org/info/rfc2315>.

[RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, DOI 10.17487/RFC2630, June 1999, <http://www.rfc-editor.org/info/rfc2630>.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, DOI 10.17487/RFC2631, June 1999, <http://www.rfc-editor.org/info/rfc2631>.

[RFC2632] Ramsdell, B., Ed., "S/MIME Version 3 Certificate Handling", RFC 2632, DOI 10.17487/RFC2632, June 1999, <http://www.rfc-editor.org/info/rfc2632>.

[RFC2633] Ramsdell, B., Ed., "S/MIME Version 3 Message Specification", RFC 2633, DOI 10.17487/RFC2633, June 1999, <http://www.rfc-editor.org/info/rfc2633>.

[RFC2785] Zuccherato, R., "Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME", RFC 2785, DOI 10.17487/RFC2785, March 2000, <http://www.rfc-editor.org/info/rfc2785>.

[RFC3218] Rescorla, E., "Preventing the Million Message Attack on Cryptographic Message Syntax", RFC 3218, DOI 10.17487/RFC3218, January 2002, <http://www.rfc-editor.org/info/rfc3218>.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, DOI 10.17487/RFC3766, April 2004, <http://www.rfc-editor.org/info/rfc3766>.

[RFC3850] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling", RFC 3850, DOI 10.17487/RFC3850, July 2004, <http://www.rfc-editor.org/info/rfc3850>.

[RFC3851] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, DOI 10.17487/RFC3851, July 2004, <http://www.rfc-editor.org/info/rfc3851>.

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, DOI 10.17487/RFC3852, July 2004, <http://www.rfc-editor.org/info/rfc3852>.

[RFC4270] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, DOI 10.17487/RFC4270, November 2005, <http://www.rfc-editor.org/info/rfc4270>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5750] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling", RFC 5750, DOI 10.17487/RFC5750, January 2010, <http://www.rfc-editor.org/info/rfc5750>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <http://www.rfc-editor.org/info/rfc5751>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <http://www.rfc-editor.org/info/rfc6151>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <http://www.rfc-editor.org/info/rfc6194>.

[RFC6278] Herzog, J. and R. Khazan, "Use of Static-Static Elliptic Curve Diffie-Hellman Key Agreement in Cryptographic Message Syntax", RFC 6278, DOI 10.17487/RFC6278, June 2011, <http://www.rfc-editor.org/info/rfc6278>.

[RFC7905] Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., and S. Josefsson, "ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)", RFC 7905, DOI 10.17487/RFC7905, June 2016, <http://www.rfc-editor.org/info/rfc7905>.

[SMIMEv2] "S/MIME version v2".

This group of documents represents S/MIME version 2. This set of documents are [RFC2311], [RFC2312], [RFC2313], [RFC2314], and [RFC2315].

[SMIMEv3] "S/MIME version 3".

This group of documents represents S/MIME version 3. This set of documents are [RFC2630], [RFC2631], [RFC2632], [RFC2633], [RFC2634], and [RFC5035].

[SMIMEv3.1]
"S/MIME version 3.1".

This group of documents represents S/MIME version 3.1. This set of documents are [RFC2634], [RFC3850], [RFC3851], [RFC3852], and [RFC5035].

[SMIMEv3.2]
"S/MIME version 3.2".

This group of documents represents S/MIME version 3.2. This set of documents are [RFC2634], [RFC5750], [RFC5751], [RFC5652], and [RFC5035].

[SP800-57]
National Institute of Standards and Technology (NIST), "Special Publication 800-57: Recommendation for Key Management", August 2005.

[TripleDES]
Tuchman, W., "Hellman Presents No Shortcut Solutions to DES"", IEEE Spectrum v. 16, n. 7, pp 40-41, July 1979.

Appendix A. ASN.1 Module

Note: The ASN.1 module contained herein is unchanged from RFC 3851 [SMIMEv3.1] with the exception of a change to the prefersBinaryInside ASN.1 comment. This module uses the 1988 version of ASN.1.

```
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 [CMS]
      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 by the 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
   }

   -- receipentKeyId is spelt incorrectly, but kept for historical
   -- reasons.
```

```
   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 OID indicates an ability to receive
   -- messages with binary encoding inside the CMS wrapper.
   -- The preferBinaryInside attribute's value field is ABSENT.

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

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

   -- Signature Algorithms Not Found in [CMSALG], [CMS-SHA2], [RSAPSS],
   -- and [RSAOAEP]

   --
   -- 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
```

Appendix B. Processing of Historic Mail

Over the course of updating the S/MIME specifications, the set of recommended algorithms has been modified each time the document has been updated. This means that if a user has historic emails and their user agent has been updated to only support the current set of recommended algorithms some of those old emails will no longer be accessible. It is strongly suggested that user agents implement some of the following algorithms for dealing with historic emails.

B.1. DigestAlgorithmIdentifier

The following algorithms have been called our for some level of support by previous S/MIME specifications:

- SHA-1 was dropped in [SMIMEv3.5]. SHA-1 is no longer considerd to be secure as it is no longer collision-resistant. The IETF statement on SHA-1 can be found in [RFC6194] but it is out-of-date relative to the most recient advances.

- MD5 was dropped in [SMIMEv3.5]. MD5 is no longer considered to be secure as it is no longer collision-resistant. Details can be found in [RFC6151].

B.2. Signature Algorithms

There are a number of problems with validating signatures on sufficently historic messages. For this reason it is strongly suggested that UAs treat these signatures differently from those on current messages. These problems include:

- CAs are not required to keep certificates on a CRL beyond one update after a certificate has expired. This means that unless CRLs are cached as part of the message it is not always possible to check if a certificate has been revoked. The same problems exist with OCSP responses as they may be based on a CRL rather than on the certificate database.

- RSA and DSA keys of less than 2048 bits are now considered by many experts to be cryptographically insecure (due to advances in computing power). Such keys were previously considered secure, so processing of historic signed messages will often result in the use of weak keys. Implementations that wish to support previous versions of S/MIME or process old messages need to consider the security risks that result from smaller key sizes (e.g., spoofed messages) versus the costs of denial of service.

  [SMIMEv3.1] set the lower limit on suggested key sizes for creating and validation at 1024 bits. Prior to that the lower bound on key sizes was 512 bits.

- Hash functions used to validate signatures on historic messages may longer be considered to be secure. (See below.) While there are not currently any known practical pre-image or second pre-image attacks against MD5 or SHA-1, the fact they are no longer considered to be collision resistent the security levels of the signatures are generally considered suspect.

- The previous two issues apply to the certificates used to validate the binding of the public key to the identity that signed the message as well.

The following algorithms have been called out for some level of support by previous S/MIME specifications:

- RSA with MD5 was dropped in [SMIMEv3.5]. MD5 is no longer considered to be secure as it is no longer collision-resistant. Details can be found in [RFC6151].

- RSA and DSA with SHA-1 were dropped in [SMIMEv3.5]. SHA-1 is no longer considered to be secure as it is no longer collision-resistant. The IETF statment on SHA-1 can be found in [RFC6194] but it is out-of-date relative to the most recent advances.

- DSA with SHA-256 was dropped in [SMIMEv3.5]. DSA has been replaced by elliptic curve versions.

Note that S/MIME v3.1 clients support verifying id-dsa-with-sha1 and rsaEncryption and might not implement sha256withRSAEncryption. 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.

B.3. ContentEncryptionAlgorithmIdentifier

The following algorithms have been called out for some level of support by previous S/MIME specifications:

- RC2/40 [RFC2268] was dropped in [SMIMEv3.2]. The algorithm is known to be insecure and, if supported, should only be used to decrypt existing email.

- DES EDE3 CBC [TripleDES], also known as "tripleDES" is dropped in [SMIMEv3.5]. This algorithms is removed from the supported list due to the fact that it has a 64-bit block size and the fact that it offers less that 128-bits of security. This algorithm should be supported only to decrypt existing email, it should not be used to encrypt new emails.

Appendix C. Moving S/MIME v2 Message Specification to Historic Status

The S/MIME v3 [SMIMEv3], v3.1 [SMIMEv3.1], and v3.2 [SMIMEv3.2] are backwards compatible with the S/MIME v2 Message Specification [SMIMEv2], with the exception of the algorithms (dropped RC2/40 requirement and added DSA and RSASSA-PSS requirements). Therefore, it is recommended that RFC 2311 [SMIMEv2] be moved to Historic status.

Appendix D. Acknowledgments

Many thanks go out to the other authors of the S/MIME version 2 Message Specification RFC: Steve Dusse, Paul Hoffman, Laurence Lundblade, and Lisa Repka. Without v2, there wouldn't be a v3, v3.1, v3.2 or v3.5.

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 because they made direct contributions to various versions of this document:

Tony Capel, Piers Chivers, Dave Crocker, Bill Flanigan, Peter Gutmann, Alfred Hoenes, Paul Hoffman, Russ Housley, William Ottaway, and John Pawling.

Authors' Addresses

Jim Schaad
August Cellars

Email: ietf@augustcellars.com

Blake Ramsdell
Brute Squad Labs, Inc.

Email: blaker@gmail.com

Sean Turner
sn3rd

Email: sean@sn3rd.com