Internet Draft
draft-dusse-smime-msg-06.txt
November 08, 1997
Expires in six months

Steve Dusse,
RSA Data Security
Paul Hoffman,
Internet Mail Consortium
Blake Ramsdell,
Worldtalk
Laurence Lundblade,
Qualcomm
Lisa Repka,
Netscape

S/MIME Message Specification

Status of this memo

1. Introduction

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

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

Further, S/MIME can be used in automated message transfer agents that use

cryptographic security services that do not require any human intervention, such as the signing of software-generated documents and the encryption of FAX messages sent over the Internet.

1.1 Specification Overview

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

This draft defines how to create a MIME body part that has been cryptographically enhanced according to PKCS #7 [PKCS-7]. This draft also defines the application/pkcs7-mime MIME type that can be used to transport those body parts. This draft also defines how to create certification requests that conform to PKCS #10 [PKCS-10], and the application/pkcs10 MIME type for transporting those requests.

This draft also discusses how to use the multipart/signed MIME type defined in [MIME-SECURE] to transport S/MIME signed messages. This draft also defines the application/pkcs7-signature MIME type, which is also used to transport S/MIME signed messages. This specification is compatible with PKCS #7 in that it uses the data types defined by PKCS #7.

In order to create S/MIME messages, an agent has to follow specifications in this draft, as well as some of the specifications listed in the following documents:

- "PKCS #1: RSA Encryption", [PKCS-1]
- "PKCS #7: Cryptographic Message Syntax", [PKCS-7]
- "PKCS #10: Certification Request Syntax", [PKCS-10]

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

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

1.2 Terminology

Throughout this draft, the terms MUST, MUST NOT, SHOULD, and SHOULD NOT are used in capital letters. This conforms to the definitions in [MUSTSHOULD].

[MUSTSHOULD] defines the use of these key words to help make the intent of standards track documents as clear as possible. The same key words are used in this document to help implementors achieve interoperability.

1.3 Definitions

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

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

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

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

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

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

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

Binary data: Arbitrary data.

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

1.4 Compatibility with Prior Practice of S/MIME

Appendix C contains important information about how S/MIME agents following this specification should act in order to have the greatest interoperability with earlier implementations of S/MIME.

1.5 Discussion of This Draft

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

```
     ietf-smime-request@imc.org
```

with the single word

```
     subscribe
```

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

2. PKCS #7 Options

The PKCS #7 message format allows for a wide variety of options in content and algorithm support. This section puts forth a number of support

requirements and recommendations in order to achieve a base level of interoperability among all S/MIME implementations.

2.1 DigestAlgorithmIdentifier

Receiving agents MUST support SHA-1 [SHA1] and MD5 [MD5].

Sending agents SHOULD use SHA-1.

2.2 DigestEncryptionAlgorithmIdentifier

Receiving agents MUST support rsaEncryption, defined in [PKCS-1]. Receiving agents MUST support verification of signatures using RSA public key sizes from 512 bits to 1024 bits.

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

2.3 KeyEncryptionAlgorithmIdentifier

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

Sending agents MUST support rsaEncryption. Sending agents MUST support encryption of symmetric keys with RSA public keys at key sizes from 512 bits to 1024 bits.

2.4 General Syntax

The PKCS #7 defines six distinct content types: "data", "signedData", "envelopedData", "signedAndEnvelopedData", "digestedData", and "encryptedData". Receiving agents MUST support the "data", "signedData" and "envelopedData" content types. Sending agents may or may not send out any of the content types, depending on the services that the agent supports.

2.4.1 Data Content Type

Sending agents MUST use the "data" content type as the content within other content types to indicate the message content which has had security services applied to it.

2.4.2 SignedData Content Type

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

2.4.3 EnvelopedData Content Type

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

2.5 Attribute SignerInfo Type

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

Receiving agents MUST be able to handle zero or one instance of each of the signed attributes described in this section.

Sending agents SHOULD be able to generate one instance of each of the signed attributes described in this section, and SHOULD include these attributes in each signed message sent.

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

2.5.1 Signing-Time Attribute

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

Sending agents MUST encode signing time through the year 2049 as UTCTime; signing times in 2050 or later MUST be encoded as GeneralizedTime. Agents MUST interpret the year field (YY) as follows: if YY is greater than or equal to 50, the year is interpreted as 19YY; if YY is less than 50, the year is interpreted as 20YY.

2.5.2 sMIMECapabilities Attribute

The sMIMECapabilities attribute includes signature algorithms (such as "md5WithRSAEncryption"), symmetric algorithms (such as "DES-CBC"), and key encipherment algorithms (such as "rsaEncryption"). It also includes a non-algorithm capability which is the preference for signedData. The sMIMECapabilities were designed to be flexible and extensible so that, in the future, a means of identifying other capabilities and preferences such as certificates can be added in a way that will not cause current clients to break.

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

their categories (signature algorithms, symmetric algorithms, key encipherment algorithms, etc.)

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

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

There is a list of OIDs (the registered sMIMECapabilities list) that is centrally maintained and is separate from this draft. The list of OIDs is maintained by the Internet Mail Consortium at <http://www.imc.org/ietf-smime/oids.html>.

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

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

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

2.6 ContentEncryptionAlgorithmIdentifier

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

Sending agents SHOULD support encryption with RC2/40 and tripleDES.

2.6.1 Deciding Which Encryption Method To Use

When a sending agent creates an encrypted message, it has to decide which type of encryption to use. The decision process involves using information

garnered from the capabilities lists included in messages received from the recipient, as well as out-of-band information such as private agreements, user preferences, legal restrictions, and so on.

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

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

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

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

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

Sections 2.6.2.1 through 2.6.2.4 describe the decisions a sending agent SHOULD use in deciding which type of encryption should be applied to a message. These rules are ordered, so the sending agent SHOULD make its decision in the order given.

2.6.2.1 Rule 1: Known Capabilities

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

2.6.2.2 Rule 2: Unknown Capabilities, Known Use of Encryption

If:
 - the sending agent has no knowledge of the encryption capabilities
   of the recipient,
 - and the sending agent has received at least one message from the
   recipient,
 - and the last encrypted message received from the recipient had a
   trusted signature on it,
then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from the recipient.

2.6.2.3 Rule 3: Unknown Capabilities, Risk of Failed Decryption

If:
 - the sending agent has no knowledge of the encryption capabilities
   of the recipient,
 - and the sending agent is willing to risk that the recipient may
   not be able to decrypt the message,
then the sending agent SHOULD use tripleDES.

2.6.2.4 Rule 4: Unknown Capabilities, No Risk of Failed Decryption

If:
 - the sending agent has no knowledge of the encryption capabilities
   of the recipient,
 - and the sending agent is not willing to risk that the recipient
   may not be able to decrypt the message,
then the sending agent MUST use RC2/40.

2.6.3 Choosing Weak Encryption

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

2.6.4 Multiple Recipients

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

3. Creating S/MIME Messages

This section describes the S/MIME message formats and how they are created.

S/MIME messages are a combination of MIME bodies and PKCS objects. Several MIME types as well as several PKCS objects are used. The data to be secured is always a canonical MIME entity. The MIME entity and other data, such as certificates and algorithm identifiers, are given to PKCS processing facilities which produces a PKCS object. The PKCS object is then finally wrapped in MIME.

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

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

3.1 Preparing the MIME Entity for Signing or Enveloping

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

The MIME entity that is secured and described in this section can be thought of as the "inside" MIME entity. That is, it is the "innermost" object in what is possibly a larger MIME message. Processing "outside" MIME entities into PKCS #7 objects is described in Section 3.2, 3.4 and elsewhere.

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

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

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

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

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

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

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

3.1.1 Canonicalization

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

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

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

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

3.1.2 Transfer Encoding

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

S/MIME implementations SHOULD however use transfer encoding described in section 3.1.3 for all MIME entities they secure. The reason for securing only 7-bit MIME entities, even for enveloped data that are not exposed to

the transport, is that it allows the MIME entity to be handled in any environment without changing it. For example, a trusted gateway might remove the envelope, but not the signature, of a message, and then forward the signed message on to the end recipient so that they can verify the signatures directly. If the transport internal to the site is not 8-bit clean, such as on a wide-area network with a single mail gateway, verifying the signature will not be possible unless the original MIME entity was only 7-bit data.

3.1.3 Transfer Encoding for Signing Using multipart/signed

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

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

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

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

3.1.4 Sample Canonical MIME Entity

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

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

```
    Content-Type: multipart/mixed; boundary=bar
```

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

=A1Hola Michael!

How do you like the new S/MIME specification?

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

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

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

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

--bar--
```

3.2 The application/pkcs7-mime Type

The application/pkcs7-mime type is used to carry PKCS #7 objects of several types including envelopedData and signedData. The details of constructing these entities is described in subsequent sections. This section describes the general characteristics of the application/pkcs7-mime type.

This MIME type always carries a single PKCS #7 object. The PKCS #7 object must always be BER encoding of the ASN.1 syntax describing the object. The contentInfo field of the carried PKCS #7 object always contains a MIME entity that is prepared as described in section 3.1. The contentInfo field must never be empty.

Since PKCS #7 objects are binary data, in most cases base-64 transfer encoding is appropriate, in particular when used with SMTP transport. The transfer encoding used depends on the transport through which the object is to be sent, and is not a characteristic of the MIME type.

Note that this discussion refers to the transfer encoding of the PKCS #7 object or "outside" MIME entity. It is completely distinct from, and unrelated to, the transfer encoding of the MIME entity secured by the PKCS #7 object, the "inside" object, which is described in section 3.1.

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

3.2.1 The name and filename Parameters

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

MIME Type	File Extension
application/pkcs7-mime (signedData, envelopedData)	.p7m
application/pkcs7-mime (degenerate signedData "certs-only" message)	.p7c
application/pkcs7-signature	.p7s
application/pkcs10	.p10

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

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

3.3 Creating an Enveloped-only Message

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

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

Step 2. The MIME entity and other required data is processed into a PKCS #7 object of type envelopedData.

Step 3. The PKCS #7 object is inserted into an application/pkcs7-mime MIME entity.

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

A sample message would be:

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

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

3.4 Creating a Signed-only Message

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

3.4.1 Choosing a Format for Signed-only Messages

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

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

Messages signed using the signedData format cannot be viewed by a recipient unless they have S/MIME facilities. However, if they have S/MIME

facilities, these messages can always be verified if they were not changed in transit.

3.4.2 Signing Using application/pkcs7-mime and SignedData

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

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

Step 2. The MIME entity and other required data is processed into a PKCS #7 object of type signedData

Step 3. The PKCS #7 object is inserted into an application/pkcs7-mime MIME entity

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

A sample message would be:

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

567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH
HUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyHh
6YT64V0GhIGfHfQbnj75
```

3.4.3 Signing Using the multipart/signed Format

This format is a clear-signing format. Recipients without any S/MIME or PKCS processing facilities are able to view the message. It makes use of the multipart/signed MIME type described in [MIME-SECURE]. The multipart/signed MIME type has two parts. The first part contains the MIME entity that is to be signed; the second part contains the signature, which is a PKCS #7 detached signature.

3.4.3.1 The application/pkcs7-signature MIME Type

This MIME type always contains a single PKCS #7 object of type signedData. The contentInfo field of the PKCS #7 object must be empty. The signerInfos field contains the signatures for the MIME entity. The details of the registered type are given in Appendix E.

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

3.4.3.2 Creating a multipart/signed Message

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

Step 2. The MIME entity is presented to PKCS #7 processing in order to obtain an object of type signedData with an empty contentInfo field.

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 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 used in the calculation of the Message Integrity Check. The value of the micalg parameter SHOULD be one of the following:

Algorithm used	Value
MD5	md5
SHA-1	sha1
any other	unknown

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

3.4.3.3 Sample multipart/signed Message

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

--boundary42
Content-Type: text/plain
```

```
    This is a clear-signed message.

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

    ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
    4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
    n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
    7GhIGfHfYT64VQbnj756

    --boundary42--
```

3.5 Signing and Encrypting

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

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

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

3.6 Creating a Certificates-only Message

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

Step 1. The certificates are made available to the PKCS #7 generating process which creates a PKCS #7 object of type signedData. The contentInfo and signerInfos fields must be empty.

Step 2. The PKCS #7 signedData object is enclosed in an application/pkcs7-mime MIME entity

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

3.7 Creating a Registration Request

A typical application which allows a user to generate cryptographic

information has to submit that information to a certification authority, who transforms it into a certificate. PKCS #10 describes a syntax for certification requests. The application/pkcs10 body type MUST be used to transfer a PKCS #10 certification request.

The details of certification requests and the process of obtaining a certificate are beyond the scope of this draft. Instead, only the format of data used in application/pkcs10 is defined.

3.7.1 Format of the application/pkcs10 Body

PKCS #10 defines the ASN.1 type CertificationRequest for use in submitting a certification request. Therefore, when the MIME content type application/pkcs10 is used, the body MUST be a CertificationRequest, encoded using the Basic Encoding Rules (BER).

Although BER is specified, instead of the more restrictive DER, a typical application will use DER since the CertificationRequest's CertificationRequestInfo has to be DER-encoded in order to be signed. A robust application SHOULD output DER, but allow BER or DER on input.

Data produced by BER or DER is 8-bit, but many transports are limited to 7-bit data. Therefore, a suitable 7-bit Content-Transfer-Encoding SHOULD be applied. The base64 Content-Transfer-Encoding SHOULD be used with application/pkcs10, although any 7-bit transfer encoding may work.

3.7.2 Sending and Receiving an application/pkcs10 Body Part

For sending a certificate-signing request, the application/pkcs10 message format MUST be used to convey a PKCS #10 certificate-signing request. Note that for sending certificates and CRLs messages without any signed content, the application/pkcs7-mime message format MUST be used to convey a degenerate PKCS #7 signedData "certs-only" message.

To send an application/pkcs10 body, the application generates the cryptographic information for the user. The details of the cryptographic information are beyond the scope of this draft.

Step 1. The cryptographic information is placed within a PKCS #10 CertificationRequest.

Step 2. The CertificationRequest is encoded according to BER or DER (typically, DER).

Step 3. As a typical step, the DER-encoded CertificationRequest is also base64 encoded so that it is 7-bit data suitable for transfer in SMTP. This then becomes the body of an application/pkcs10 body part.

The result might look like this:

```
Content-Type: application/pkcs10; name=smime.p10
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p10

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

A typical application only needs to send a certification request. It is a certification authority that has to receive and process the request. The steps for recovering the CertificationRequest from the message are straightforward but are not presented here. The procedures for processing the certification request are beyond the scope of this document.

3.8 Identifying an S/MIME Message

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

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

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

MIME type:   application/pkcs10
parameters:  any
file suffix: any

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

MIME type:   application/octet-stream
parameters:  any
file suffix: p7m, p7s, aps, p7c, p10
```

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 draft does not cover how S/MIME agents handle certificates, only what they do after a certificate has been validated or rejected. S/MIME

certification issues are covered in a different document.

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

An S/MIME agent or some related administrative utility or function MUST be capable of generating RSA key pairs on behalf of the user. Each key pair MUST be generated from a good source of non-deterministic random input and protected in a secure fashion.

A user agent SHOULD generate RSA key pairs at a minimum key size of 768 bits and a maximum key size of 1024 bits. A user agent MUST NOT generate RSA key pairs less than 512 bits long. Some agents created in the United States have chosen to create 512 bit keys in order to get more advantageous export licenses. However, 512 bit keys are considered by many to be cryptographically insecure.

Implementors should be aware that multiple (active) key pairs may be associated with a single individual. For example, one key pair may be used to support confidentiality, while a different key pair may be used for authentication.

5. Security Considerations

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

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

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

the time, and therefore this draft gives a framework for using those estimates in choosing algorithms.

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

A. Object Identifiers and Syntax

The syntax for SMIMECapability is:

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

sMIMECapabilities ::= SEQUENCE OF SMIMECapability
```

A.1 Content Encryption Algorithms

```
RC2-CBC OBJECT IDENTIFIER ::=
     {iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) 2}
```

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

```
RC2-CBC parameter ::=  SEQUENCE {
     rc2ParameterVersion  INTEGER,
     iv                   OCTET STRING (8)}
```

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

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

For DES-CBC and DES-EDE3-CBC, the parameter should be encoded as:

```
CBCParameter :: IV

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

A.2 Digest Algorithms

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

sha-1 OBJECT IDENTIFIER ::=
```

```
     {iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 26}
```

A.3 Asymmetric Encryption Algorithms

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

rsa OBJECT IDENTIFIER ::=
     {joint-iso-ccitt(2) ds(5) algorithm(8) encryptionAlgorithm(1) 1}
```

A.4 Signature Algorithms

```
md2WithRSAEncryption OBJECT IDENTIFIER ::=
     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 2}

md5WithRSAEncryption OBJECT IDENTIFIER ::=
     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 4}

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

A.5 Signed Attributes

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

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

B. References

[3DES] W. Tuchman, "Hellman Presents No Shortcut Solutions To DES," IEEE Spectrum, v. 16, n. 7, July 1979, pp40-41.

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

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

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

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

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

[MIME-SECURE] "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847

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

[PKCS-1] "PKCS #1: RSA Encryption", Internet Draft draft-hoffman-pkcs-rsa-encrypt

[PKCS-7] "PKCS #7: Cryptographic Message Syntax", Internet Draft draft-hoffman-pkcs-crypt-msg

[PKCS-10] "PKCS #10: Certification Request Syntax", Internet Draft draft-hoffman-pkcs-certif-req

[RC2] "Description of the RC2 Encryption Algorithm", Internet Draft draft-rivest-rc2desc

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

C. Compatibility with Prior Practice in S/MIME

S/MIME was originally developed by RSA Data Security, Inc. Many developers implemented S/MIME agents before this document was published. All S/MIME receiving agents SHOULD make every attempt to interoperate with these earlier implementations of S/MIME.

C.1 Early MIME Types

Some early implementations of S/MIME agents used the following MIME types:

application/x-pkcs7-mime
application/x-pkcs7-signature
application/x-pkcs10

In each case, the "x-" subtypes correspond to the subtypes described in this document without the "x-".

C.2 Profiles

Early S/MIME documentation had two profiles for encryption: "restricted" and "unrestricted". The difference between these profiles historically came about due to US Government export regulations, as described at the end of this section. It is expected that in the future, there will be few agents that only use the restricted profile.

Briefly, the restricted profile required the ability to encrypt and decrypt using RSA's trade-secret RC2 algorithm in CBC mode with 40-bit keys. The

unrestricted profile required the ability to encrypt and decrypt using RSA's trade-secret RC2 algorithm in CBC mode with 40-bit keys, and to encrypt and decrypt using tripleDES. The restricted profile also had non-mandatory suggestions for other algorithms, but these were not widely implemented.

It is important to note that many current implementations of S/MIME use the restricted profile.

C.2.1 Historical Reasons for the Existence of Two Encryption Profiles

Due to US Government export regulations, an S/MIME agent which supports a strong content encryption algorithm such as DES would not be freely exportable outside of North America. US software manufacturers have been compelled to incorporate an exportable or "restricted" content encryption algorithm in order to create a widely exportable version of their product. S/MIME agents created in the US and intended for US domestic use (or use under special State Department export licenses) can utilize stronger, "unrestricted" content encryption. However, in order to achieve interoperability, such agents need to support whatever exportable algorithm is incorporated in restricted S/MIME agents.

The RC2 symmetric encryption algorithm has been approved by the US Government for "expedited" export licensing at certain key sizes. Consequently, support for the RC2 algorithm in CBC mode is required for baseline interoperability in all S/MIME implementations. Support for other strong symmetric encryption algorithms such as RC5 CBC, DES CBC and DES EDE3-CBC for content encryption is strongly encouraged where possible.

D. Revision History

The following changes were made between the -05 and -06 revisions of this draft:

Removed discussion of "application/mime" wrapping because no one has implemented it and because the specification for application/mime is in flux. This entailed removing section 3.4.3.4, a bit of the table near the end of section 3.8, and text throughout appendix F.

Changed the case of SMIMECapabilities to sMIMECapabilities everywhere.

Changed the references for ASN.1, BER, and DER back to their 1988 documents.

Fixed error in the MIME examples in 3.1.4 (left off the C-T-E).

Removed antique text from first paragraph 3.5.

Fixed section numbering in Appendix A.

In A.1, changed "other than 32" to "greater than 32".

Removed "smime-type" from E.2 and E.3, where they appeared by mistake.

E. Request for New MIME Subtypes

E.1 application/pkcs7-mime

To: ietf-types@iana.org
Subject: Registration of MIME media type application/pkcs7-mime

MIME media type name: application

MIME subtype name: pkcs7-mime

Required parameters: none

Optional parameters: name, filename, smime-type

Encoding considerations: Will be binary data, therefore should use base64 encoding

Security considerations: Described in [PKCS-7]

Interoperability considerations: Designed to carry data formatted with PKCS-7, as described in [PKCS-7]

Published specification: draft-dusse-smime-msg-xx

Applications which use this media type: Secure Internet mail and other secure data transports.

Additional information:
File extension(s): .p7m and .p7c
Macintosh File Type Code(s):

Person & email address to contact for further information:
Steve Dusse, spock@rsa.com

Intended usage: COMMON

E.2 application/pkcs7-signature

To: ietf-types@iana.org
Subject: Registration of MIME media type application/pkcs7-signature

MIME media type name: application

MIME subtype name: pkcs7-signature

Required parameters: none

Optional parameters: name, filename

Encoding considerations: Will be binary data, therefore should use base64 encoding

Security considerations: Described in [PKCS-7]

Interoperability considerations: Designed to carry digital signatures with PKCS-7, as described in [PKCS-7]

Published specification: draft-dusse-smime-msg-xx

Applications which use this media type: Secure Internet mail and other secure data transports.

Additional information:
File extension(s): .p7s
Macintosh File Type Code(s):

Person & email address to contact for further information:
Steve Dusse, spock@rsa.com

Intended usage: COMMON

E.3 application/pkcs10

To: ietf-types@iana.org
Subject: Registration of MIME media type application/pkcs10

MIME media type name: application

MIME subtype name: pkcs10

Required parameters: none

Optional parameters: name, filename

Encoding considerations: Will be binary data, therefore should use base64 encoding

Security considerations: Described in [PKCS-10]

Interoperability considerations: Designed to carry digital certificates formatted with PKCS-10, as described in [PKCS-10]

Published specification: draft-dusse-smime-msg-xx

Applications which use this media type: Secure Internet mail and other transports where certificates are required.

Additional information:
File extension(s): .p10
Macintosh File Type Code(s):

Person & email address to contact for further information:
Steve Dusse, spock@rsa.com

Intended usage: COMMON

F. Encapsulating Signed Messages for Internet Transport

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

Ideally, the multipart/signed format would be the only format used because it provides a truly backwards compatible way to sign MIME entities. In a pure MIME environment with very capable user agents, this would be possible. The world, however, is more complex than this.

One problem with the multipart/signed format occurs with gateways to non-MIME environments. In these environments, the gateway will generally not be S/MIME aware, will not recognize the multipart/signed type, and will default its treatment to multipart/mixed as is prescribed by the MIME standard. The real problem occurs when the gateway also applies conversions to the MIME structure of the original message that is being signed and is contained in the first part of the multipart/signed structure, such as the gateway converting text and attachments to the local format. Because the signature is over the MIME structure of the original message, but the original message is now decomposed and transformed, the signature cannot be verified. Because MIME encoding of a particular set of body parts can be done in many different ways, there is no way to reconstruct the original MIME entity over which the signature was computed.

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

F.1 Solutions to the Problem

To work around these two problems, the application/pkcs7-mime type can be used. When going through a gateway, it will be defaulted to the MIME type of application/octet-stream and treated as a single opaque entity. That is,

the message will be treated as an attachment of unknown type, converted into the local representation for an attachment and thus can be made available to an S/MIME facility completely intact. A similar result is achieved when a user agent similarly treats the application/pkcs7-mime MIME entity as a simple leaf node of the MIME structure and makes it available to viewer applications.

Another way to work around these problems is to encapsulate the multipart/signed MIME entity in a MIME entity that will not be damaged by the gateway. At the time that this draft is being written, there is a proposal for a MIME entity "application/mime" for this purpose. However, no implementations of S/MIME use this type of wrapping.

F.2 Encapsulation in an Non-MIME Environment

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

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

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

G. Acknowledgements

Significant contributions to the content of this draft were made by many people, including Jeff Thompson and Jeff Weinstein.

H. Authors' addresses

Steve Dusse
RSA Data Security, Inc.
100 Marine Parkway, #500
Redwood City, CA 94065 USA
(415) 595-8782
spock@rsa.com

Paul Hoffman
Internet Mail Consortium

127 Segre Place
Santa Cruz, CA 95060
(408) 426-9827
phoffman@imc.org

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

Laurence Lundblade
QUALCOMM Incorporated
Eudora Division
6455 Lusk Boulevard
San Diego, California 92121-2779
(800) 238-3672
lgl@qualcomm.com

Lisa Repka
Netscape Communications Corporation
501 East Middlefield Road
Mountain View, CA 94043
(415) 254-1900
repka@netscape.com