Internet Draft
draft-ietf-smime-ess-00.txt
November 18, 1997
Expires in six months

Editor: Paul Hoffman
Internet Mail Consortium

Enhanced Security Services for S/MIME

Status of this memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

1. Introduction

This document describes three optional security service extensions for S/MIME. These services provide functionality that is similar to the Message Security Protocol [MSP], but are useful in many other environments, particularly business and finance. The services are:

- signed receipts
- security labels
- secure mailing lists

The services described here are extensions to S/MIME version 2 [SMIME2] and S/MIME version 3 [SMIME3]. Most of this document can be used with S/MIME version 2, which relies on PKCS #7 version 1.5 [PKCS7-1.5]. A small number of the services require mechanisms described in Cryptographic Message Syntax [CMS].

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/>.

1.1 Triple Wrapping

Some of the features of each service use the concept of a "triple wrapped" message. A triple wrapped message is one that has been signed, then encrypted, then signed again. The signers of the inner and outer signatures may be different entities or the same entity. Note that the S/MIME specification does not limit the number of nested encapsulations, so there may be more than three wrappings.

1.1.1 Purpose of Triple Wrapping

Not all messages need to be triple wrapped. Triple wrapping is used when a signed and encrypted message must be signed, then encrypted, and then processed by other agents that have to be authenticated by the final recipient.

The inside signature is used for content integrity, non-repudiation with proof of origin, and binding attributes (such as a security label) to the original content. These attributes go from the originator to the recipient, regardless of the number of intermediate entities such as mail list agents that process the message. The authenticated attributes can be used for access control to the inner body. Requests for signed receipts by the originator are carried in the inside signature as well.

The encrypted body provides confidentiality, including confidentiality of the attributes that are carried in the inside signature.

The outside signature provides authentication and integrity for information that is processed hop-by-hop, where each hop is an intermediate entity such as a mail list agent. The outer signature binds attributes (such as a security label) to the encrypted body. These attributes can be used for access control and routing decisions.

1.1.2 Steps for Triple Wrapping

The steps to create a triple wrapped message are:

1. Start with a message body, called the "original content".

2. Encapsulate the original content with the appropriate MIME headers. An exception to this MIME encapsulation rule is that a signed receipt is not put in MIME headers.

3. Sign the result of step 2 (the MIME headers and the original content), turning it into a application/pkcs7-mime body part, and add the appropriate MIME headers. The application/pkcs7-mime body part is called the "inside signature".

4. Encrypt the result of step 3 (the MIME headers and the inside signature) as a single block, turning it into another (larger) application/pkcs7-mime body part, and add the appropriate MIME headers. The application/pkcs7-mime body part is called the "encrypted body".

5. Sign the result of step 4 (the MIME headers and the encrypted body) as a single block, turning it into another (even larger) application/pkcs7-mime body part, and add the appropriate MIME headers. The application/pkcs7-mime body part is called the "outside signature".

6. The result of step 5 (the MIME headers and the outside signature) is the triple wrapped message.

1.2 Format of a Triple Wrapped Message

A triple wrapped message has eight layers of encapsulation. Starting from the innermost layer and working outwards, the layers are:

```
Original content ("Hello, world!")
MIME entity
ContentInfo: data type
Inner SignedData block
MIME entity
ContentInfo: data type
EnvelopedData block
MIME entity
ContentInfo: data type
Outer SignedData block
MIME entity
```

Note that both the inner and outer signed blocks use the SignedData construct of S/MIME. As defined in [PKCS7-1.5] and [CMS], each SignedData and EnvelopedData object MUST be encapsulated by a ContentInfo SEQUENCE. There is no purpose to use the multipart/signed format in inner case because it is known that the recipient is known to be able to process S/MIME messages (because they decrypted the middle wrapper). There may be a purpose in using multipart/signed in the outer layer, but only so that a non-S/MIME agent could see that the next inner layer is encrypted. However, this is not of great value, since all it shows the recipient is that he or she wouldn't have been able to read the message anyways.

1.3 Security Services and Triple Wrapping

The three security services described in this document are used with triple wrapped messages in different ways. This section briefly describes the relationship of each service with triple wrapping; the other sections of the document go into greater detail.

1.3.1 Signed Receipts and Triple Wrapping

A signed receipt may be requested in any SignedData object. However, if a signed receipt is requested for a triple wrapped message, the receipt request MUST be in the inside signature, not in the outside signature. A secure mailing list agent may change the receipt policy in the outside signature of a triple wrapped message when that message is processed by the mailing list.

Note: the signed receipts and receipt requests described in this draft differ from those described in the work done by the IETF Receipt Notification Working Group. The output of that Working Group, when finished, is not expected to work well with triple wrapped messages as described in this document.

1.3.2 Security Labels and Triple Wrapping

A security label may be included in the authenticated attributes of a SignedData object. A security label attribute may be included in either the inner signature, outer signature, or both.

The inner security label is used for access control decisions related to the plaintext original content. The inner signature provides authentication and cryptographically protects the original signer's security label that is on the inside body. This strategy facilitates the forwarding of messages because the original signer's security label is included in the SignedData block which can be forwarded to a third party that can verify the inner signature which will cover the inner security label. The confidentiality security service can be applied to the inner security label by encrypting the entire inner SignedData block within an EnvelopedData block.

A security label may also be included in the authenticated attributes of the outer SignedData block which will include the sensitivities of the encrypted message. The outer security label is used for access control and routing decisions related to the encrypted message. Note that a security label attribute can only be used in an authenticatedAttributes block. A securityLabel attribute MUST NOT be used in an EnvelopedData or unauthenticated attributes.

1.3.3 Secure Mailing Lists and Triple Wrapping

Secure mail list message processing depends on the structure of S/MIME layers present in the message sent to the mail list agent. The agent never changes the data that was hashed to form the inner signature, if such a signature is present. If an outer signature is present, then the agent will modify the data that was hashed to form that outer signature. In all cases, the agent adds or updates an mlExpansionHistory attribute to document the agent's processing, and ultimately adds or replaces the outer signature on the message to be distributed.

1.3.4 Placement of Attributes

Certain attributes should be placed in the inner or outer SignedData message; some attributes can be in either. Further, some attributes must be authenticated, while authentication is optional for others. The following table summarizes the recommendation of this profile.

Attribute	Inner or outer	MUST BE authenticated
contentHints	either	no
contentIdentifier	either	no

contentType	either	no
counterSignature	either	no
encapsulatedContentType	either	no
messageDigest	either	yes
mlExpansionHistory	outer only	yes
receiptRequest	inner only	yes
signingTime	either	yes
smimeCapabilities	either	yes
securityLabel	either	yes

Note that the inner and outer signatures are for different senders, so that the same attribute in the two signatures could lead to very different consequences.

ContentIdentifier is an attribute (OCTET STRING) used to carry a unique identifier assigned to the message. EncapsulatedContentType is an attribute used to carry the content type of the encapsulated content. These attributes are needed in addition to the fields carried in the receiptRequest attribute.

1.4 Object Identifiers

The object identifiers for many of the objects described in this draft are found in the registry kept at <http://www.imc.org/ietf-smime/oids.html>. When this draft moves to standards track within the IETF, it is intended that the IANA will maintain this registry.

2. Signed Receipts

Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message. This receipt is bound to the original message through the signature; consequently, this service may be requested only if a message is signed. The receipt sender may optionally also encrypt a receipt to provide confidentiality between the receipt sender and the receipt recipient.

2.1 Signed Receipt Concepts

The originator of a message may request a signed receipt from the message's recipients. The request is indicated by adding a receiptRequest attribute to the authenticatedAttributes field of the SignerInfo object for which the receipt is requested. The receiving user agent software SHOULD automatically create a signed receipt when requested to do so, and return the receipt in accordance with mailing list expansion options, local security policies, and configuration options.

Because receipts involve the interaction of two parties, the terminology can sometimes be confusing. In this section, the "sender" is the agent that sent the original message that included a request for a receipt. The "receiver" is the party that received that message and generated the

receipt.

The steps in a typical transaction are:

1. Sender creates a signed message including a receipt request attribute (Section 2.2).

2. Sender transmits the resulting message to the recipient or recipients.

3. Recipient receives message and determines if there is a valid signature and receipt request in the message (Section 2.3).

4. Recipient creates a signed receipt (Section 2.4).

5. Recipient transmits the resulting signed receipt message to the sender (Section 2.5).

6. Sender receives the message and validates that it contains a signed receipt for the original message (Section 2.6). This validation relies on the sender having kept a digest value of the original message (Section 2.7) or a copy of the original message.

The ASN.1 syntax for the receipt request is given in Section 2.8; the ASN.1 syntax for the receipt is given in Section 2.9.

Note that an agent SHOULD remember when it has sent a receipt so that it can avoid re-sending a receipt each time it processes the message.

2.2 Receipt Request Creation

Multi-layer S/MIME messages may contain multiple SignedData layers. However, receipts may be requested only for the innermost SignedData layer in a multi-layer S/MIME message, such as a triple wrapped message. Only one receiptRequest attribute can be included in the authenticatedAttributes of a SignerInfo. A sender requests receipts by placing a receiptRequest attribute in the authenticated attributes of a signerInfo as follows:

1. A receiptRequest data structure is created.

2. The encapsulated content type is optionally noted in the encapsulatedContentType field.

3. A signed content identifier for the message is created and assigned to the signedContentIdentifier field. The signedContentIdentifier is used to associate the signed receipt with the message requesting the signed receipt.

4. The entities requested to return a signed receipt are noted in the receiptsFrom field.

5. If receipts are to be returned to entities other than or in addition to the message originator, a list of receipt recipients is assigned to the

receiptsTo field. The originator's name(s) MUST be included in the receiptsTo list if receipt recipients in addition to the originator are requested.

6. The completed receiptRequest attribute is placed in the authenticatedAttributes field of the SignerInfo object.

2.2.1 Multiple Receipt Requests

There can be multiple SignerInfos within a SignedData object, and each SignerInfo may include authenticatedAttributes. Therefore, a single SignedData object may include multiple SignerInfos, each SignerInfo having a receiptRequest attribute. For example, an originator can send a signed message with two SignerInfos, one containing a DSS signature, the other containing an RSA signature.

Each recipient SHOULD return only one signed receipt.

Not all of the SignerInfos need to include receipt requests, but in all of the SignerInfos that do conatin receipt requests, the receipt requests MUST be identical.

2.3 Receipt Request Processing

A receiptRequest is associated only with the SignerInfo object in which the receipt request attribute is directly attached. Processing software SHOULD examine the authenticatedAttributes field of each of the SignerInfos for which it verifies a signature in the innermost signedData object to determine if a receipt is requested. This may result in the receiving agent processing multiple receiptRequest attributes included in a single SignedData object. Because all receiptRequest attributes in a SignedData object must be identical, the receiving application fully processes (as described in the following paragraphs) the first receiptRequest that it encounters in a SignerInfo that it can verify, and it then ensures that all other receiptRequests are identical to the first one encountered.

If a receiptRequest attribute is absent from the authenticated attributes, then a signed receipt has not been requested from any of the message recipients and MUST NOT be created. If a receiptRequest attribute is present in the authenticated attributes, then a signed receipt has been requested from some or all of the message recipients. Note that in some cases, a receiving agent might receive two almost-identical messages, one with a receipt request and the other without one. In this case, the receiving agent may choose whether or not to send a receipt.

If a receiptRequest attribute is present in the authenticated attributes, the following process SHOULD be used to determine if a message recipient has been requested to return a signed receipt.

1. If an mlExpansionHistory attribute is present in the outermost signedData block, do one of the following two steps, based on the absence or presence of mlReceiptPolicy:

1.1. If an mlReceiptPolicy value is absent from the last MLData element, a Mail List receipt policy has not been specified and the processing software SHOULD examine the receiptRequest attribute value to determine if a receipt should be created and returned.

1.2. If an mlReceiptPolicy value is present in the last MLData element, do one of the following two steps, based on the value of mlReceiptPolicy:

1.2.1. If the mlReceiptPolicy value is none, then the receipt policy of the Mail List supersedes the originator's request for a signed receipt and a signed receipt MUST NOT be created.

1.2.2. If the mlReceiptPolicy value is insteadOf or inAdditionTo, the processing software SHOULD examine the receiptsFrom value from the receiptRequest attribute to determine if a receipt should be created and returned. If a receipt is created, the insteadOf and inAdditionTo fields identify entities that SHOULD be sent the receipt instead of or in addition to the originator.

2. If the receiptsFrom value of the receiptRequest attribute is allOrNone, do one of the following three steps based on the value of allOrNone.

2.1. If the value of allOrNone is noReceipt, then a signed receipt MUST NOT be created.

2.2. If the value of allOrNone is allReceipts, then a signed receipt SHOULD be created.

2.3. If the value of allOrNone is firstTierRecipients, do one of the following two steps based on the presence of an mlExpansionHistory attribute:

2.3.1. If an mlExpansionHistory attribute is present, then this recipient is not a first tier recipient and a signed receipt MUST NOT be created.

2.3.2. If an mlExpansionHistory attribute is not present, then a signed receipt SHOULD be created.

3. If the receiptsFrom value of the receiptRequest attribute is a receiptList:

3.1. If receiptList contains one of the GeneralNames of the recipient, then a signed receipt should be created.

3.2. If receiptList does not contain one of the GeneralNames of the recipient, then a signed receipt MUST NOT be created.

A flow chart for the above steps to be executed for each signerInfo for which the receiving agent verifies the signature would be:

0. Receipt Request attribute present?
YES -> 1.
NO -> STOP
1. Has mlExpansionHistory?
YES -> 1.1.
NO -> 2.
1.1. mlReceiptPolicy absent?
YES -> examine receiptRequest, then -> 2.
NO -> 1.2.
1.2. Pick based on value of mlReceiptPolicy.
none -> 1.2.1.
insteadOf or inAdditionTo -> 1.2.2.
1.2.1. Use ML's policy, then -> STOP
1.2.2. Examine receiptsFrom for name, determine if a receipt should be created, create it if required, then -> STOP.
2. Is value of receiptsFrom allOrNone.
YES -> Pick based on value of allOrNone.
noReceipt -> 2.1.
allReceipts -> 2.2.
firstTierRecipients -> 2.3.
NO -> 3.
2.1. STOP.
2.2. Create a receipt, then -> STOP.
2.3. Has mlExpansionHistory?
YES -> 2.3.1.
NO -> 2.3.2.
2.3.1. STOP.
2.3.2. Create a receipt, then -> STOP.
3. Is receiptsFrom value of receiptRequest a receiptList?
YES -> 3.1.
NO -> STOP.
3.1. Does receiptList contain the recipient?
YES -> Create a receipt, then -> STOP.
NO -> 3.2.
3.2. STOP.

2.4 Receipt Creation

A signed receipt is created as follows:

1. The signature of the original message is validated. A receipt MUST NOT be created for a message with an invalid signature.

2. A Receipt structure is created.

2.1. The value of the version field is set to 1.

2.2. The encapsulatedContentType and signedContentIdentifier

values are copied from the SignerInfo's receiptRequest attribute to the corresponding fields in the Receipt structure.

2.3. The signatureValue (i.e. digital signature or MAC) from the original message SignerInfo structure is copied to the originatorSignatureValue field in the receipt structure.

3. The Receipt structure is ASN.1 DER encoded to produce a data stream, D1.

4. D1 is concatenated to the end of the ASN.1 encoded original message content to produce a data stream, D2. The "ASN.1 encoded original message content" is the data composing the SignedData contentInfo content ANY. For example, if SignedData is being used to sign MIME-encapsulated data, then the signedData ContentInfo content ANY field will include a Data content type (i.e. OCTET STRING). In that case, the "ASN.1 encoded original message content" is the DER encoded value of the Data OCTET STRING. The Data OCTET STRING tag and length octets are not included in the hashing.

5. D2 is digested to produce a digest value, H2, for the receipt.

6. The Receipt structure MUST be directly included within a SignedData structure using H2 as the message digest to be signed. This results in a single ASN.1 encoded object composed of a SignedData including the Receipt content type. The Receipt MUST NOT be encapsulated in a MIME header or any other header prior to being encoded as part of the SignedData object.

6.1. A contentHints attribute is created and SHOULD be added to the SignerInfo structure's authenticated attributes.

The signed message that contains the signed receipt SHOULD have a signingTime attribute so that the recipient knows when the receipt was created.

2.5 Determining the Recipients of the Signed Receipt

If a signed receipt was created by the process described in the sections above, then the software MUST use the following process to determine to whom the signed receipt should be sent.

1. If the receiptsTo is present in the Receipt Request attribute, then the software initiates the sequence of recipients with the value(s) of receiptsTo; otherwise, the software initiates the sequence of recipients with the signer (that is, the originator) of the SignerInfo that includes the Receipt Request attribute.

2. If the MlExpansionHistory attribute is present in the outer SignedData block, and the last MLData contains an MLReceiptPolicy value of insteadOf, then the software replaces the sequence of recipients with the value(s) of insteadOf.

3. If the MlExpansionHistory attribute is present in the outer SignedData block and the last MLData contains an MLReceiptPolicy value of

inAdditionTo, then the software adds the value(s) of inAdditionTo to the sequence of recipients.

2.6 Receipt Validation

A receipt is validated as follows:

1. The signed receipt is communicated as a single ASN.1 encoded object composed of a SignedData directly including a Receipt content type. ASN.1 decode the signedData object including the Receipt.

2. Retrieve the encapsulatedContentType and signedContentIdentifier values from the Receipt data structure to identify the message being receipted.

3. Acquire H2 based on the message identification information.

3.1. If H2 has been saved locally, it must be located and retrieved.

3.2. If H2 has not been saved, the original message must be located and H2 must be recreated from the original message and related information as described in the "Receipt Digest Value" section.

4. Obtain the alleged receipt signature value from the receipt's signatureValue field and validate the signature using the retrieved signature value and H2, the calculated hash value.

[More is needed here or in an appendix detailing how to do this for each kind of signature.]

2.7 Receipt Digest Value

The requester of a signed receipt must retain either the message for which a receipt is being requested, or a receipt digest value (hash value) derived from the message for later receipt validation. Retaining the digest value usually requires less local storage than retaining a message because digest values typically contain fewer bytes than the messages they are derived from.

Message content and message identity information are used to calculate a receipt digest value as follows:

1. The encapsulated content type, signed content identifier, and encrypted digest value (signature value) derived from the message content are copied from the SignerInfo including the receiptRequest into a Receipt structure. The Receipt structure version field is set to 1.

2. The Receipt structure is ASN.1 DER encoded to produce a data stream, D1, as described in the "Receipt Creation" section.

3. D1 is concatenated to the end of the ASN.1 encoded original message

content to produce a data stream, D2.

4. D2 is digested to produce a digest value, H2, for the receipt.

2.8 Receipt Request Syntax

A receiptRequest attribute value has ASN.1 type receiptRequest. Use the receiptRequest attribute only within the authenticated attributes associated with a signed message.

```
receiptRequest ::= SEQUENCE {
    encapsulatedContentType        EncapsulatedContentType OPTIONAL,
    signedContentIdentifier        ContentIdentifier,
    receiptsFrom                   ReceiptsFrom,
    receiptsTo                     SEQUENCE (SIZE (1..ub-receiptsTo))
                                       OF GeneralNames OPTIONAL }
ub-receiptsTo INTEGER ::= 16
```

The encapsulatedContentType field identifies the content type of the original message. In BuiltinContentType, the values of 0 and 1 have been deprecated and SHOULD NOT be used.

```
EncapsulatedContentType ::= CHOICE {
    built-in                       BuiltinContentType,
    external                       ExternalContentType,
    externalWithSubtype            ExternalContentWithSubtype }

BuiltinContentType ::= [APPLICATION 6] INTEGER {
    unidentified                   (0),
    external                       (1),
    interpersonal-messaging-1984   (2),
    interpersonal-messaging-1988   (22),
    edi-messaging (35),
    voice-messaging (40)} (0..ub-built-in-content-type)
ub-built-in-content-type INTEGER ::= 32767

ExternalContentType ::= OBJECT IDENTIFIER

ExternalContentWithSubtype ::= SEQUENCE {
    external        ExternalContentType,
    subtype        INTEGER }
```

A signedContentIdentifier MUST be created by the message originator when creating a receipt request. To ensure global uniqueness, the minimal signedContentIdentifier SHOULD contain a concatenation of user-specific identification information (such as a user name or public keying material identification information), a GeneralizedTime string, and a random number.

The receiptsFrom field is used by the originator to specify the recipients requested to return a signed receipt. A CHOICE is provided to allow specification of:

- no receipts are requested

```
 - receipts from all recipients are requested
 - receipts from first tier (recipients that did not receive the
   message as members of a mailing list) recipients are requested
 - receipts from a specific list of recipients are requested

ReceiptsFrom ::= CHOICE {
    allOrNone          [0] AllOrNone,
    receiptList        [1] SEQUENCE OF GeneralNames }

AllOrNone ::= INTEGER {
    noReceipt              (0),
    allReceipts            (1),
    firstTierRecipients    (2) }
```

The receiptsTo field is used by the originator to identify the user(s) to whom the identified recipient should send signed receipts. Use the field only if receipts must be sent to users other than, or in addition to, the originator. If the receiptsTo field is used to designate recipients in addition to the originator, then the originator's name(s) MUST be included in the receiptsTo list.

2.9 Receipt Syntax

Receipts are represented using a new content type, receipt. The receipt content type shall have ASN.1 type Receipt. Receipts must be encapsulated within a SignedData message.

```
Receipt := SEQUENCE {
    version                         Version,
    encapsulatedContentType         EncapsulatedContentType OPTIONAL,
    signedContentIdentifier         OCTET STRING,
    originatorSignatureValue        OCTET STRING }
```

The version field defines the syntax version number, which is 1 for this version of the standard.

The encapsulatedContentType and signedContentIdentifier fields are copied from the receiptRequest attribute of the SignerInfo contained within the message being receipted, and are used to link the receipt to the original signed message. The originatorSignatureValue field contains the signatureValue copied from the SignerInfo requesting the signed receipt.

2.10 Content Hints

Many applications find it useful to have information that describes the innermost signed content of a multi-layer message available on the outermost signature layer. The contentHints attribute provides such information.

Content-hints attribute values have ASN.1 type contentHints.

```
contentHints ::= SEQUENCE {
```

```
    contentDescription    DirectoryString [ ub-conDesc } OPTIONAL,
    receipt               BOOLEAN DEFAULT FALSE }

ub-conDesc INTEGER ::= 128

DirectoryString { INTEGER : maxSize } ::= CHOICE {
    teletexString      TeletexString (SIZE (1..maxSize)),
    printableString    PrintableString (SIZE (1..maxSize)),
    bmpString          BMPString (SIZE (1..maxSize)),
    universalString    UniversalString (SIZE (1..maxSize)) }
```

Messages that contain a signed receipt MUST include this attribute with the receipt value set to TRUE. The contentDescription field may be used to provide information that the recipient may use to select protected messages for processing, such as a message subject.

3. Security Labels

This section describes the syntax to be used for security labels that can optionally be associated with S/MIME encapsulated data. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation.

"Authorization" is the act of granting rights and/or privileges to users permitting them access to an object. "Access control" is a means of enforcing these authorizations. The sensitivity information in a security label can be compared with a user's authorizations to determine if the user is allowed to access the content that is protected by S/MIME encapsulation.

Security labels may be used for other purposes such as a source of routing information. The labels are often priority based ("secret", "confidential", "restricted", and so on) or role-based, describing which kind of people can see the information ("patient's health-care team", "medical billing agents", "unrestricted", and so on).

3.1 Security Label Processing Rules

A sending agent may include a security label attribute in the authenticated attributes of a signedData object. A receiving agent examines the security label on a recevied message and determines whether or not the recipinet is allowed to see the contents of the message.

3.1.1 Adding Security Labels

A sending agent that is using security labels MUST put the security label attribute in the authenticatedAttributes field of a SignerInfo block. The security label attribute MUST NOT be included in the unauthenticated attributes. Integrity and authentication security services MUST be applied to the security label, therefore it MUST be included as an authenticated attribute, if used. This causes the security label attribute to be part of the data that is hashed to form the SignerInfo signature value. A

SignerInfo block MUST NOT have more than one security label authenticated attribute.

When there are multiple SignedData blocks applied to a message, a security label attribute may be included in either the inner signature, outer signature, or both. A security label authenticated attribute may be included in a authenticatedAttributes field within the inner SignedData block. The inner security label will include the sensitivities of the original content and will be used for access control decisions related to the plaintext encapsulated content. The inner signature provides authentication of the inner security label and cryptographically protects the original signer's inner security label of the original content.

When the originator signs the plaintext content and authenticated attributes, the inner security label is bound to the plaintext content. An intermediate entity cannot change the inner security label without invalidating the inner signature. The confidentiality security service can be applied to the inner security label by encrypting the entire inner signedData object within an EnvelopedData block.

A security label authenticated attribute may also be included in a authenticatedAttributes field within the outer SignedData block. The outer security label will include the sensitivities of the encrypted message and will be used for access control decisions related to the encrypted message and for routing decisions. The outer signature provides authentication of the outer security label (as well as for the encapsulated content which may include nested S/MIME messages).

There can be multiple SignerInfos within a SignedData object, and each SignerInfo may include authenticatedAttributes. Therefore, a single SignedData object may include multiple security labels, each SignerInfo having a securityLabel attribute. For example, an originator can send a signed message with two SignerInfos, one containing a DSS signature, the other containing an RSA signature. Not all of the SignerInfos need to include security labels, but in all of the SignerInfos that do conatin security labels, the security labels MUST be identical.

A recipient SHOULD process a securityLabel attribute only if the recipient can verify the signature of the SignerInfo which covers the securityLabel attribute. A recipient SHOULD NOT use a security label that the recipient cannot authenticate.

3.1.2 Processing Security Labels

A receiving agent that processes security labels MUST process the securityLabel attribute, if present, in each SignerInfo in the SignedData object for which it verifies the signature. This may result in the receiving agent processing multiple security labels included in a single SignedData object. Because all security labels in a SignedData object must be identical, the receiving application processes (such as performing access control) on the first securityLabel that it encounters in a

SignerInfo that it can verify, and then ensures that all other securityLabels are identical to the first one encountered.

A receiving agent that processes security labels SHOULD have a local policy about whether or not to show the inner content of an incoming messages that has a security label with a security policy identifier that the processing software does not recognize. If the receiving agent does not recognize the securityLabel security-policy-identifier value, it SHOULD stop processing the message and indicate an error.

3.2 Syntax of securityLabel

The securityLabel syntax is copied directly from [MTSABS] ASN.1 module. (The MTSAbstractService module begins with "DEFINITIONS IMPLICIT TAGS ::=".) Further, the securityLabel syntax is identical to that used in [MSP4] and [ACP120].

```
securityLabel ::= SET {
     security-policy-identifier   SecurityPolicyIdentifier OPTIONAL,
     security-classification      SecurityClassification OPTIONAL,
     privacy-mark                 PrivacyMark OPTIONAL,
     security-categories          SecurityCategories OPTIONAL }

SecurityPolicyIdentifier ::= OBJECT IDENTIFIER

SecurityClassification ::= INTEGER {
  unmarked      (0),
  unclassified  (1),
  restricted    (2),
  confidential  (3),
  secret        (4),
  top-secret    (5) } (0..ub-integer-options)

ub-integer-options INTEGER ::= 256

PrivacyMark ::= PrintableString (SIZE (1..ub-privacy-mark-length))

ub-privacy-mark-length INTEGER ::= 128

SecurityCategories ::= SET SIZE (1..ub-security-categories) OF
                            SecurityCategory

ub-security-categories INTEGER ::= 64

SecurityCategory ::= SEQUENCE {
    type      [0] OBJECT IDENTIFIER,
    value     [1] ANY -- defined by type}

-Note: The aforementioned SecurityCategory syntax produces identical
-hex encodings as the following SecurityCategory syntax that is
-documented in the X.411 specification:
-
```

```
-SecurityCategory ::= SEQUENCE {
-     type                      [0]  SECURITY-CATEGORY,
-     value                     [1]  ANY DEFINED BY type }
-
-SECURITY-CATEGORY MACRO ::=
-BEGIN
-TYPE NOTATION ::= type | empty
-VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
-END
```

3.3 Security Label Components

This section gives more detail on the the various components of the securityLabel syntax.

3.3.1 Security Policy Identifier

A security policy is a set of criteria for the provision of security services. The securityLabel security-policy-identifier is used to identify the security policy in force to which the security label relates. It indicates the semantics of the other security label components. Even though the securityLabel security-policy-identifier is an optional field, all security labels used with S/MIME messages MUST include the security-policy-identifier.

3.3.2 Security Classification

This specification defines the use of the Security Classification field exactly as is specified in the X.411 Recommendation, which states in part:

> If present, a security-classification may have one of a hierarchical list of values. The basic security-classification hierarchy is defined in this Recommendation, but the use of these values is defined by the security-policy in force. Additional values of security-classification, and their position in the hierarchy, may also be defined by a security-policy as a local matter or by bilateral agreement. The basic security-classification hierarchy is, in ascending order: unmarked, unclassified, restricted, confidential, secret, top-secret.

This means that the security policy in force (identified by the securityLabel security-policy-identifier) defines the SecurityClassification integer values and their meanings.

An organization can develop its own security policy that defines the SecurityClassification INTEGER values and their meanings. However, the general interpretation of the X.411 specification is that the values of 0 thru 5 are reserved for the "basic hierarchy" values of unmarked, unclassified, restricted, confidential, secret, and top-secret. Note that **X.411 does not provide the rules for how these values are used to label** data and how access control is performed using these values.

There is no universal definition of the rules for using these "basic

hierarchy" values. Each organization (or group of organizations) will define a security policy which documents how the "basic hierarchy" values are used (if at all) and how access control is enforced (if at all) within their domain.

Therefore, the security-classification value MUST be accompanied by a security-policy-identifier value to define the rules for its use. For example, a company's "secret" classification may convey a different meaning than the US Government "secret" classification. In summary, a security policy SHOULD NOT use integers 0 through 5 for other than their X.411 meanings, and SHOULD instead use other values in a hierarchical fashion.

Note that the set of valid security-classification values MUST be hierarchical, but these values do not necessarily need to be in ascending numerical order. Further, the values do not need to be contiguous.

For example, in the Defense Message System 1.0 security policy, the security-classification value of 11 indicates Sensitive-But-Unclassified and 5 indicates top-secret. The hierarchy of sensistivity ranks top-secret as more sensitive than Sensitive-But-Unclassified even though the numerical value of top-secret is less than Sensitive-But-Unclassified.

(Of course, if security-classification values are both hierarchical and in ascending order, a casual reader of the security policy is more likely to understand it.)

An example of a security policy that does not use any of the X.411 values might be:
10 -- anyone
15 -- Morgan Corporation and its contractors
20 -- Morgan Corporation employees
25 -- Morgan Corporation board of directors

An example of a security policy that uses part of the X.411 hierarchy might be:
0 -- unmarked
1 -- unclassified, can be read by everyone
2 -- restricted to Timberwolf Productions staff
6 -- can only be read to Timberwolf Productions executives

3.3.3 Privacy Mark

If present, the securityLabel privacy-mark is not used for access control. The content of the securityLabel privacy-mark may be defined by the security policy in force (identified by the securityLabel security-policy-identifier) which may define a list of values to be used. Alternately, the value may be determined by the originator of the security-label.

3.3.4 Security Categories

If present, the securityLabel security-categories provide further

granularity for the sensitivity of the message. The security policy in force (identified by the securityLabel security-policy-identifier) is used to indicate the syntaxes that are allowed to be present in the securityLabel security-categories. Alternately, the security-categories and their values may be defined by bilateral agreement.

4. Mail List Management

Sending agents must create recipient-specific data structures for each recipient of an encrypted message. This process can impair performance for messages sent to a large number of recipients. Thus, Mail List Agents (MLAs) that can take a single message and perform the recipient-specific encryption for every recipient are often desired.

An MLA appears to the message originator as a normal message recipient, but the MLA acts as a message expansion point for a Mail List (ML). The sender of a message directs the message to the MLA, which then redistributes the message to the members of the ML. This process offloads the per-recipient processing from individual user agents and allows for more efficient management of large MLs. MLs are true message recipients served by MLAs that provide cryptographic and expansion services for the mailing list.

In addition to cryptographic handling of messages, secure mailing lists also have to prevent mail loops. A mail loop is where one mailing list is a member of a second mailing list, and the second mailing list is a member of the first. A message will go from one list to the other in a rapidly-cascading sucession of mail that will be distributed to all other members of boths lists.

To prevent mail loops, MLAs use the mlExpansionHistory attribute of the outer signature of a triple wrapped message. The mlExpansionHistory attribute is essentially a list of every MLA that has processed the message. If an MLA sees its own unique entity identifier in the list, it knows that a loop has been formed, and does not send the message to the list again.

4.1 Mail List Expansion

Mail list expansion processing is noted in the value of the mlExpansionHistory attribute, located in the authenticated attributes of the MLA's SignerInfo block. The MLA creates or updates the authenticated mlExpansionHistory attribute value each time the MLA expands and signs a message for members of a mail list.

The MLA MUST add an MLData record containing the MLA's identification information, date and time of expansion, and optional receipt policy to the end of the mail list expansion history sequence. If the mlExpansionHistory attribute is absent, then the MLA MUST add the attribute and the current expansion becomes the first element of the sequence. If the mlExpansionHistory attribute is present, then the MLA MUST add the current expansion information to the end of the existing MLExpansionHistory

sequence. Only one mlExpansionHistory attribute can be included in the authenticatedAttributes of a SignerInfo.

Note that if the mlExpansionHistory attribute is absent, then the recipient is a first tier message recipient.

There can be multiple SignerInfos within a SignedData object, and each SignerInfo may include authenticatedAttributes. Therefore, a single SignedData object may include multiple SignerInfos, each SignerInfo having a mlExpansionHistory attribute. For example, an originator can send a signed message with two SignerInfos, one containing a DSS signature, the other containing an RSA signature. Not all of the SignerInfos need to include mlExpansionHistory attributes, but in all of the SignerInfos that do conatin mlExpansionHistory attributes, the mlExpansionHistory attributes MUST be identical.

A recipient SHOULD only process an mlExpansionHistory attribute if the recipient can verify the signature of the SignerInfo which covers the attribute. A recipient SHOULD NOT use an mlExpansionHistory attribute which the recipient cannot authenticate.

When receiving a message that includes an outer SignedData object, a receiving agent that processes mlExpansionHistory attributes MUST process the mlExpansionHistory attribute, if present, in each SignerInfo in the SignedData object for which it verifies the signature. This may result in the receiving agent processing multiple mlExpansionHistory attributes included in a single SignedData object. Because all mlExpansionHistory attributes must be identical, the receiving application processes the first mlExpansionHistory attribute that it encounters in a SignerInfo that it can verify, and then ensures that all other mlExpansionHistory attributes are identical to the first one encountered.

4.1.1 Detecting Mail List Expansion Loops

Prior to expanding a message, the MLA examines the value of any existing mail list expansion history attribute to detect an expansion loop. An expansion loop exists when a message expanded by a specific MLA for a specific mail list is redelivered to the same MLA for the same mail list.

Expansion loops are detected by examining the mailListIdentifier field of each MLData entry found in the mail list expansion history. If an MLA finds its own identification information, then the MLA must discontinue expansion processing and should provide warning of an expansion loop to a human mail list administrator. The mail list administrator is responsible for correcting the loop condition.

4.2 Mail List Agent Processing

MLA message processing depends on the structure of S/MIME layers found in the processed message. In all cases, the MLA ultimately signs the message and adds or updates an mlExpansionHistory attribute to document MLA

processing. In all cases, the MLA may need to perform access control before distributing the message to mail list members if the message contains a SignedData block and an associated securityLabel attribute. If a securityLabel authenticated attribute is used for access control, then the signature of the signerInfo block including the securityLabel authenticated attribute MUST be verified before using the security label. The MLA should continue parsing the MIME-encapsulated message to determine if there is a security label associated with an encapsulated SignedData object. This may include decrypting an EnvelopedData object to determine if an encapsulated SignedData object includes a securityLabel attribute.

Each MLA that processes the message creates its own mlExpansionHistory and adds it to the sequence of mlExpansionHistory attributes already in the message. An MLA MUST NOT modify the mlExpansionHistory created by a MLA that previously processed the message. Each MLA copies the sequence of mlExpansionHistory attributes created by the MLAs that previously processed the message into the newly constructed expanded message, and adds its own mlExpansionHistory as the last element of the sequence.

The processing used depends on the type of the outermost layer of the message. There are three cases for the type of the outermost data:

- EnvelopedData
- SignedData
- data

4.2.1 Processing for EnvelopedData

1. The MLA locates its own RecipientInfo and uses the information it contains to obtain the message key.

2. The MLA removes the existing recipientInfos field and replaces it with a new recipientInfos value built from RecipientInfo structures created for each member of the mailing list.

3. The MLA encapsulates the expanded encrypted message in a SignedData block, adding an mlExpansionHistory attribute as described in the "Mail List Expansion" section to document the expansion.

4. The MLA signs the new message and delivers the updated message to mail list members to complete MLA processing.

4.2.2 Processing for SignedData

MLA processing of multi-layer messages depends on the type of data in each of the layers. Step 3 below specifies that different processing will take place depending on the type of PKCS #7 message that has been signed. That is, it needs to know the type of data at the next inner layer, which may or may not be the innermost layer.

1. The MLA verifies the signature value found in the outermost SignedData layer associated with the signed data. MLA processing of the message terminates if the message signature is invalid.

2. If the outermost SignedData layer includes an authenticated mlExpansionHistory attribute the MLA checks for an expansion loop as described in the "Detecting Mail List Expansion Loops" section.

3. Determine the type of the data that has been signed. That is, look at the type of data on the layer just below the SignedData, which may or may not be the "innermost" layer. Based on the type of data, perform either step 3.1 (EnvelopedData), step 3.2 (SignedData), or step 3.3 (all other types).

3.1. If the signed data is EnvelopedData, the MLA performs expansion processing of the encrypted message as described previously. Note that this process invalidates the signature value in the outermost SignedData layer associated with the original encrypted message. Proceed to section 3.2 with the result of the expansion.

3.2. If the signed data is SignedData, or is the result of expanding an EnvelopedData block in step 3.1:

3.2.1. The MLA strips the existing outermost SignedData layer after remembering the value of the mlExpansionHistory attribute in that layer, if one was there.

3.2.2. If the signed data is EnvelopedData (from step 3.1), the MLA encapsulates the expanded encrypted message in a new outermost SignedData layer. On the other hand, if the signed data is SignedData (from step 3.2), the MLA encapsulates the signed data in a new outermost SignedData layer.

3.2.3. The MLA adds an mlExpansionHistory attribute. The SignedData layer created by the MLA replaces the original outermost SignedData layer.

3.2.3.1. If the original outermost SignedData layer included an mlExpansionHistory attribute, the attribute's value is copied and updated with the current ML expansion information as described in the "Mail List Expansion" section.

3.2.3.2. If the original outermost SignedData layer did not include an mlExpansionHistory attribute, a new attribute value is created with the current ML expansion information as described in the "Mail List Expansion" section.

3.3. If the signed data is not EnvelopedData or SignedData:

3.3.1. The MLA encapsulates the received signedData object in an SignedData object, and adds an mlExpansionHistory attribute to the outer SignedData object containing the current ML expansion information as described in the "Mail List Expansion" section.

4. The MLA signs the new message and delivers the updated message to mail list members to complete MLA processing.

A flow chart for the above steps would be:

1. Has a valid signature?
YES -> 2.
NO -> STOP.
2. Does outermost SignedData layer
contain mlExpansionHistory?
YES -> Check it, then -> 3.
NO -> 3.
3. Check type of data just below outermost
SignedData.
EnvelopedData -> 3.1.
SignedData -> 3.2.
all others -> 3.3.
3.1. Expand the encrypted message, then -> 3.2.
3.2. -> 3.2.1.
3.2.1. Strip outermost SignedData layer, note value of
mlExpansionHistory, then -> 3.2.2.
3.2.2. Encapsulate in new signature, then -> 3.2.3.
3.2.3. Add mlExpansionHistory. Was there an old mlExpansionHistory?
YES -> copy the old mlExpansionHistory values, then -> 4.
NO -> create new mlExpansionHistory value, then -> 4.
3.3. Is the signed data EnvelopedData or SignedData?
YES -> 4.
NO -> Encapsulate in a SignedData layer and add a mlExpansionHistory attribute.
4. Sign message, deliver it, STOP.

4.2.3 Processing for data

1. The MLA encapsulates the message in a SignedData layer, and adds an mlExpansionHistory attribute containing the current ML expansion information as described in the "Mail List Expansion" section.

2. The MLA signs the new message and delivers the updated message to mail list members to complete MLA processing.

4.3 Mail List Expansion History Syntax

An mlExpansionHistory attribute value has ASN.1 type MLExpansionHistory. If there are more than ub-ml-expansion-hsitory mailing lists in the sequence, the processing agent should return an error.

```
MLExpansionHistory ::= SEQUENCE (SIZE (1..ub-ml-expansion-history))
                           OF MLData
ub-ml-expansion-history INTEGER ::= 64
```

MLData contains the expansion history describing each MLA that has processed a message. As an MLA distributes a message to members of an ML, the MLA records its unique identifier, date and time of expansion, and receipt policy in an MLData structure.

```
MLData ::= SEQUENCE {
    mailListIdentifier  EntityIdentifier,
    expansionTime       GeneralizedTime,
    mlReceiptPolicy     MLReceiptPolicy OPTIONAL }

EntityIdentifier ::= CHOICE {
  issuerAndSerialNumber  IssuerAndSerialNumber, -- From PKCS #7
  subjectKeyIdentifier   KeyIdentifier }

KeyIdentifier ::= OCTET STRING
```

The receipt policy of the ML can withdraw the originator's request for the return of a signed receipt. However, if the originator of the message has not requested a signed receipt, the MLA cannot request a signed receipt.

When present, the mlReceiptPolicy specifies a receipt policy that supersedes the originator's request for signed receipts. The policy can be one of three possibilities: receipts MUST NOT be returned (none); receipts should be returned to an alternate list of recipients, instead of to the originator (insteadOf); or receipts should be returned to a list of recipients in addition to the originator (inAdditionTo).

```
MLReceiptPolicy ::= CHOICE {
    none            [0] NULL,
    insteadOf       [1] SEQUENCE (SIZE (1..ub-insteadOf))
                          OF GeneralNames,
    inAdditionTo    [2] SEQUENCE (SIZE (1..ub-inAdditionTo))
                          OF GeneralNames }
ub-insteadOf INTEGER ::= 16
ub-inAdditionTo INTEGER ::= 16
```

5. Security Considerations

This entire document discusses security.

A. References

[ACP120] 28 Oct 97 Final Draft Allied Communication Publication (ACP) 120 Communication Security Protocol (CSP) Specification.

[CMS] Cryptographic Message Syntax, Internet Draft draft-ietf-smime-cms-xx.

[MSP4] Secure Data Network System (SDNS) Message Security Protocol (MSP) 4.0, Specification SDN.701, Revision A, 1997-02-06.

[MTSABS] 1988 International Telecommunication Union (ITU) Data Communication Networks Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures, Volume VIII, Fascicle VIII.7, Recommendation X.411; MTSAbstractService {joint-iso-ccitt mhs-motis(6) mts(3) modules(0) mts-abstract-service(1)}

[PKCS7-1.5] "PKCS #7: Cryptographic Message Syntax", Internet Draft draft-hoffman-pkcs-crypt-msg-xx.

[SMIME2] "S/MIME Version 2 Message Specification", Internet Draft draft-dusse-smime-msg-xx, and "S/MIME Version 2 Certificate Handling", Internet Draft draft-dusse-smime-cert-xx.

[SMIME3] "S/MIME Version 3 Message Specification", Internet Draft draft-ietf-smime-msg-xx, and "S/MIME Version 3 Certificate Handling", Internet Draft draft-ietf-smime-cert-xx.

B. Acknowledgements

The first draft of this work was prepared by David Solo. John Pawling did a huge amount of very detailed revision work during the many phases of the document.

Many other people have contributed hard work to this draft, including:
Bengt Ackzell
Blake Ramsdell
Carlisle Adams
Jim Schaad
Phillip Griffin
Russ Housley
Scott Hollenbeck
Steve Dusse

C. Open Issues

There is a desire for a single ASN.1 module that collects all the ASN.1 from the whole document.

There is apparently two ASN.1:1994 types (BMPString and UniversalString) in the draft, leading to invalid ASN.1.

2.4: Includes hashing the authenticatedAttributes included in the SignerInfo containing the receipt signature value included in the SignedData/Receipt. Also state that a SignedData/Receipt is not allowed to include receiptRequest or MLExpansionHistory attributes.

2.6: Examples are needed.

3.2: An OID for the securityLabel attribute is needed.

5: The security considerations section needs to be fleshed out, including discussions of what happens if receiving clients don't check things very well.

D. Changes from Draft -00 to Draft -01

Changed the file name of the draft from "draft-hoffman-smime-ess" to "draft-ietf-smime-ess".

Made the following capitalization changes throughout:
ContentHints -> contentHints
ReceiptRequest -> receiptRequest
SecurityLabel -> securityLabel

1.1.1: removed last sentence of first paragraph.

1.2: Added to the last paragraph: As defined in [PKCS7-1.5] and [CMS], each SignedData and EnvelopedData object MUST be encapsulated by a ContentInfo SEQUENCE.

1.3.1: Changed "A signed receipt may be requested in any signed body part." to "...any SignedData object."

1.3.2: Changed the beginning of the first sentence from "A security label in authenticated attributes may also be included in the outer SignedData block..." to "A security label may also be included in the authenticated attributes of the outer SignedData block...".

1.3.3: Changed last sentence to "In all cases, the agent adds or updates an mlExpansionHistory attribute to document the agent's processing, and ultimately adds or replaces the outer signature on the message to be distributed."

1.3.4: Changed "SignerInfo" to "SignedData" in the first sentence.

1.3.4: Changed ContentIdentifier to contentIdentifier and EncapsulatedContentType to encapsulatedContentType to reference the to-be-defined OIDs. Also alphabatized the table and added:

counterSignature	either	no
contentType	either	no
messageDigest	either	yes

1.4: Added this as a new section. Also, throughout the draft, removed definitions of OIDs in this draft that are actually on the OIDs page at IMC.

2.2: Added to the first paragraph: "Only one receiptRequest attribute can

be included in the authenticatedAttributes of a SignerInfo." Also changed "signed message" to "SignerInfo" in the last sentence.

2.2.1: This section is new and adds new functionality from the previous draft. It should be read carefully, and additional wordsmithing is encouraged.

2.3: Made large additions to the first paragraph. In the first bullet, changed "outermost authenticatedAttributes block" to "outermost signedData block". Also changed the lead-in paragraph to the flow chart. Also fixed **2.3.1 and 2.3.2 in the flow chart to match the text.**

2.4: Changed bullet 2.2 to have the values copied from the "SignerInfo's receiptRequest" instead of the "original message's receiptRequest". In bullet 2.3, changed "protectionValue" to "signatureValue".

2.6: Bullet 4, changed "protectionValue" to "signatureValue".

2.7: Changed the first sentence in bullet 1 to read "The encapsulated content type, signed content identifier, and encrypted digest value (signature value) derived from the message content are copied from the SignerInfo including the receiptRequest into a Receipt structure." Added the reference to the receipt creation section in bullet 3.

2.8: Change the definition of signedContentIdentifier from OCTET STRING to ContentIdentifier.

2.9: Replaced the last paragraph with better wording.

2.10: Changed the defintion of ContentHints to include { ub-conDesc }

3.1: Changed "signed message" to "signedData object" in the first paragraph.

3.1.1: In the first paragraph, changed "SignedData" to "SignerInfo". In the third paragraph, changed "signed message" to "signedData object". Also added long paragraphs at the end of this section describing multiple SignerInfos and what to do with them; this is new material that should be carefully scrutinized.

3.1.2: The section "Processing Security Labels" was also called 3.1.1 in the previous draft; renumbered it. Also, changed and added most of the text of the section.

3.2: Added the second sentence of the first paragraph, which was moved from **Appendix A. Also fixed the ub-xxx ASN.1 defintions to include INTEGER.**

4.1: Added to the end of the second paragraph: "Only one MLExpansionHistory attribute can be included in the authenticatedAttributes of a SignerInfo." Also added all the text starting with "There can be multiple....", which describes how to handle multiple SignerInfos and what to do with them. This is new material and should be checked carefully.

4.2: In the first paragraph, changed "signedData block" to "signerInfo block" and added the last two sentences. Added definitions of EntityIdentifier and KeyIdentifier.

4.2.1: Bullet 1, removed "record". Bullet 3.3.1, fixed the wording to be more accruate.

4.2.3: Removed "digestedData".

A: Updated the reference for [ACP120]. Moved the sentence from [MTSABS] to Section 3.2.

B: Gave John Pawling more direct credit for all his hard work.

E. Editor's Address

Paul Hoffman
Internet Mail Consortium
127 Segre Place
Santa Cruz, CA 95060
(408) 426-9827
phoffman@imc.org