S/MIME Working Group
Internet Draft
Category: Standards Track

J Schaad
Soaring Hawk Consulting

August 2004

Enhanced Security Services for S/MIME
draft-ietf-smime-rfc2634-update-00.txt

Status of this Memo

Abstract

This document describes the structures and procedures necessary to provide a number of additional security services for S/MIME. These services are:

- signed receipts
- security labels
- secure mailing lists
- signing certificate validation

These services can be used by any CMS (Cryptographic Message Syntax) based protocol.

This document currently only contains the sections of RFC 2634 that are being updated. The two documents will be folded together at a later date.

****

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 signed, while signing is optional for others, and some attributes must not be signed. ESS defines several types of attributes. ContentHints and ContentIdentifier MAY appear in any list of attributes. contentReference, equivalentLabel, eSSSecurityLabel and mlExpansionHistory MUST be carried in a SignedAttributes or AuthAttributes type, and MUST NOT be carried in a UnsignedAttributes, UnauthAttributes or UnprotectedAttributes type. msgSigDigest, receiptRequest and signingCertificate MUST be carried in a SignedAttributes, and MUST NOT be carried in a AuthAttributes, UnsignedAttributes, UnauthAttributes or UnprotectedAttributes type.

The following table summarizes the recommendation of this profile. In the OID column, [ESS] indicates that the attribute is defined in this document.

Attribute	OID	Inner or outer	Signed
contentHints	id-aa-contentHint [ESS]	either	MAY
contentIdentifier	id-aa-contentIdentifier [ESS]	either	MAY
contentReference	id-aa-contentReference [ESS]	either	MUST
contentType	id-contentType [CMS]	either	MUST
counterSignature	id-countersignature [CMS]	either	MUST NOT
equivalentLabel	id-aa-equivalentLabels [ESS]	either	MUST
eSSSecurityLabel	id-aa-securityLabel [ESS]	either	MUST
messageDigest	id-messageDigest [CMS]	either	MUST
msgSigDigest	id-aa-msgSigDigest [ESS]	inner only	MUST
mlExpansionHistory	id-aa-mlExpandHistory [ESS]	outer only	MUST
receiptRequest	id-aa-receiptRequest [ESS]	inner only	MUST
signingCertificate	id-aa-signingCertificate [ESS]	either	MUST
signingTime	id-signingTime [CMS]	either	MUST
smimeCapabilities	sMIMECapabilities [MSG]	either	MUST
sMIMEEncryption-			

```
  KeyPreference   |id-aa-encrypKeyPref [MSG]     |either    |MUST
mlaExpandHistory  |id-aa-mlaExpandHistory [ESS]  |outer only|MUST
receiptModify     |id-aa-receiptModify [ESS]     |either    |MUST
```

CMS defines signedAttrs as a SET OF Attribute and defines unsignedAttrs as a SET OF Attribute. ESS defines the contentHints, contentIdentifier, eSSecurityLabel, msgSigDigest, mlExpansionHistory, receiptRequest, contentReference, equivalentLabels and

signingCertificate attribute types. A signerInfo MUST NOT include multiple instances of any of the attribute types defined in ESS. Later sections of ESS specify further restrictions that apply to the receiptRequest, mlExpansionHistory and eSSecurityLabel attribute types.

CMS defines the syntax for the signed and unsigned attributes as "attrValues SET OF AttributeValue". For all of the attribute types defined in ESS, if the attribute type is present in a signerInfo, then it MUST only include a single instance of AttributeValue. In other words, there MUST NOT be zero, or multiple, instances of AttributeValue present in the attrValues SET OF AttributeValue.

If a counterSignature attribute is present, then it MUST be included in the unsigned attributes. It MUST NOT be included in the signed attributes. The only attributes that are allowed in a counterSignature attribute are counterSignature, messageDigest, signingTime, and signingCertificate.

Note that the inner and outer signatures are usually those of different senders. Because of this, 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.

2. Signed Receipts

Returning a signed receipt provides to the originator proof of delivery of a message, and allows the originator to demonstrate to a third party that the recipient was able to verify the signature of the original 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 can request a signed receipt from the message's recipients. The request is indicated by adding a receiptRequest attribute to the signedAttrs 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 retained either a copy of the original message or information extracted from the original message.

The ASN.1 syntax for the receipt request is given in Section 2.7; the ASN.1 syntax for the receipt is given in Section 2.9.

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

A receipt request can indicate that receipts be sent to many places, not just to the sender (in fact, the receipt request might indicate that the receipts should not even go to the sender). In order to

verify a receipt, the recipient of the receipt needs to be the originator or a recipient of the original message. Thus, the sender SHOULD NOT request that receipts be sent to anyone who does not have an exact copy of the message.

2.2 Receipt Request Creation

Multi-layer S/MIME messages can contain multiple SignedData layers. However, only one layer can contain a receipt request. This will generally be the innermost layer, but in some workflow applications it can be a middle or outer layer. Receipt processing MUST NOT start before all layers of CMS content are unwound so that only the innermost receipt request is processed. Only one receiptRequest attribute can be included in the signedAttrs of a SignerInfo.

A ReceiptRequest attribute MUST NOT be included in the attributes of a SignerInfo in a SignedData object that encapsulates a content type of Receipt (id-ct-receipt). In other words, the receiving agents MUST NOT request a signed receipt for a signed receipt.

A sender requests receipts by placing a receiptRequest attribute in the signed attributes of a signerInfo as follows:

1. A receiptRequest data structure is created.

2. 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.

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

4. The message originator MUST populate the receiptsTo field with a GeneralNames for each entity to whom the recipient should send the signed receipt. If the message originator wants the recipient to send the signed receipt to the originator, then the originator MUST include a GeneralNames for itself in the receiptsTo field. GeneralNames is a SEQUENCE OF GeneralName. receiptsTo is a SEQUENCE OF GeneralNames in which each GeneralNames represents an entity. There can be multiple GeneralName instances in each GeneralNames. At a minimum, the message originator MUST populate each entity's GeneralNames with the address to which the signed receipt is suppose to be sent. Optionally, the message originator MAY also populate each entity's GeneralNames with other GeneralName instances (such as directoryName).

5. The completed receiptRequest attribute is placed in the signedAttrs field of the SignerInfo object.

2.2.1 Multiple Receipt Requests

There can be multiple SignerInfos within a SignedData object, and each SignerInfo can include signedAttrs. Therefore, a single SignedData object can 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 within a SignedData object need to include receipt requests, but in all of the SignerInfos that do contain receipt requests, the receipt requests MUST be identical.

2.2.2 Information Needed to Validate Signed Receipts

The sending agent MUST retain one or both of the following items to support the validation of signed receipts returned by the recipients.

- the original SignedData object requesting the signed receipt

- the content identifier in the receipt request, the message signature digest value and the content type and signature value included in the original SignedData object. If signed receipts are requested from multiple recipients, then retaining these values is a performance enhancement because the sending agent can reuse the saved values when verifying each returned signed receipt.

2.3 Receipt Request Processing

A receiptRequest is associated only with the SignerInfo object that the receipt request is an authenticated attribute of. The behavior for processing of a receiptRequest is modified by the presence of a either a receiptPolicy or an mlaExpandHistory attribute either in the same SignerData or in a outer SignerData object.

Before processing a receiptRequest signedAttribute, the receiving agent MUST verify the following conditions:

1. The signature of the SignerInfo that covers the receiptRequest attribute MUST validate.

2. All receiptRequests for SignerInfo objects in the current SignedData object MUST be the same. (Since the attributes are DER encoded, this check can be done by a binary compare of the attributes.)

3. The encapsulated content of the message MUST NOT contain a SignedData for which a receiptRequest exists.

4. The inner-most encapsulated content of the message MUST NOT be id-ct-receipt.

A receipt MUST NOT be created of any of these conditions are not met.

If a receiptRequest attribute is absent from the signed 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 signed 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 SHOULD send a signed receipt for the message that requests a signed receipt.

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

1. If a receiptPolicy attribute is present in the SignedData block, do one of the following two steps value of ReceiptPolicy:

1.1. If the ReceiptPolicy value is none, then the receipt policy supersedes the originator's request for a signed receipt and a signed receipt MUST NOT be created.

1.2. If the ReceiptPolicy 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 allOrFirstTier, do one of the following two steps based on the value of allOrFirstTier.

2.1. If the value of allOrFirstTier is allReceipts, then a signed receipt SHOULD be created.

2.2. If the value of allOrFirstTier is firstTierRecipients, do one of the following two steps based on the presence of an mlaExpandHistory attribute in an outer SignedData block:

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

2.2.2. If an mlaExpandHistory 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. Does an outer SignedData layer exist?
       YES -> 1.1.
       NO  -> 4.
1.1. Make next SignedData layer out the current layer.
2. Current layer has a receiptPolicy attribute?
       YES -> 2.1.
       NO  -> 3.
2.1. Modify receiptsTo based on ReceiptPolicy
```



```
2.2. Go to 3.
3. Current layer has an mlaExpandHistory attribute?
       YES -> 3.1
       NO  -> 1.
3.1. Is value of receiptsFrom allOrFirstTier?
       YES -> Pick based on value of allOrFirstTier.
            allReceipts -> 1.
            firstTierReceipts -> 3.2.
       NO  -> 1.
```

3.2. Set receiptsFrom to none.
3.3. Go to 1.
4. Is receiptsFrom value a receiptList?
YES -> 4.1.
NO -> 4.2.
4.1. Does receipList contain the recipient?
YES -> 4.2.
NO -> STOP.
4.2. Create a receipt.
4.3. STOP.

2.4 Signed Receipt Creation

A signed receipt is a SignedData object encapsulating a Receipt content (also called a "SignedData/Receipt"). Signed receipts are created as follows:

1. The signature of the original SignedData signerInfo that includes the receiptRequest signed attribute MUST be successfully verified before creating the SignedData/Receipt.

   1.1. The content of the original SignedData object is digested as described in [CMS]. The resulting digest value is then compared with the value of the messageDigest attribute included in the signedAttrs of the original SignedData signerInfo. If these digest values are different, then the signature verification process fails and the SignedData/Receipt MUST NOT be created.

   1.2. The ASN.1 DER encoded signedAttrs (including messageDigest, receiptRequest and, possibly, other signed attributes) in the original SignedData signerInfo are digested as described in [CMS]. The resulting digest value, called msgSigDigest, is then used to verify the signature of the original SignedData signerInfo. If the signature verification fails, then the SignedData/Receipt MUST NOT be created.

2. A Receipt structure is created.

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

   2.2. The object identifier from the contentType attribute included in the original SignedData SignerInfo that includes the

receiptRequest attribute is copied into the Receipt contentType.

2.3. The original SignedData signerInfo receiptRequest signedContentIdentifier is copied into the Receipt signedContentIdentifier.

2.4. The signature value from the original SignedData signerInfo that includes the receiptRequest attribute is copied into the Receipt originatorSignatureValue.

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

4. D1 is digested. The resulting digest value is included as the messageDigest attribute in the signedAttrs of the SignerInfo which will eventually contain the SignedData/Receipt signature value.

5. The digest value (msgSigDigest) calculated in Step 1 to verify the signature of the original SignedData SignerInfo is included as the msgSigDigest attribute in the signedAttrs of a SignerInfo which will eventually contain the SignedData/Receipt signature value.

6. A contentType attribute including the id-ct-receipt object identifier MUST be created and added to the signed attributes of the signerInfo which will eventually contain the SignedData/Receipt signature value.

7. A signingTime attribute indicating the time that the SignedData/Receipt is signed SHOULD be created and added to the signed attributes of the SignerInfo which will eventually contain the SignedData/Receipt signature value. Other attributes (except receiptRequest) can be added to the signedAttrs of the SignerInfo.

8. The signedAttrs (messageDigest, msgSigDigest, contentType, and possibly others) of the SignerInfo are ASN.1 DER encoded and digested as described in [CMS]. The resulting digest value is used to calculate the signature value which is then included in the SignedData/Receipt signerInfo.

9. The ASN.1 DER encoded Receipt content MUST be directly encoded within the SignedData EncapContentInfo.eContent OCTET STRING defined in [CMS]. The id-ct-receipt object identifier MUST be included in the SignedData EncapContentInfo.eContentType. This results in a single ASN.1 encoded object composed of a SignedData including the Receipt content. The Data content type MUST NOT be used. The Receipt content MUST NOT be encapsulated in a MIME header or any other header prior to being encoded as part of the SignedData object.

10. The SignedData/Receipt is then put in an application/pkcs7-mime MIME wrapper with the smime-type parameter set to "signed-receipt". This will allow for identification of signed receipts

without having to crack the ASN.1 body. The smime-type parameter would still be set as normal in any layer wrapped around this message.

11. If the SignedData/Receipt is to be encrypted within an EnvelopedData object, then an outer SignedData object MUST be created that encapsulates the EnvelopedData object, and a contentHints attribute with contentType set to the id-ct-receipt object identifier MUST be included in the outer SignedData SignerInfo signedAttrs. When a receiving agent processes the outer SignedData object, the presence of the id-ct-receipt OID in the contentHints contentType indicates that a SignedData/Receipt is encrypted within the EnvelopedData object encapsulated by the outer SignedData.

All sending agents that support the generation of ESS signed receipts MUST provide the ability to send encrypted signed receipts (that is, a SignedData/Receipt encapsulated within an EnvelopedData). The sending agent MAY send an encrypted signed receipt in response to an EnvelopedData-encapsulated SignedData requesting a signed receipt. It is a matter of local policy regarding whether or not the signed receipt should be encrypted. The ESS signed receipt includes the message digest value calculated for the original SignedData object that requested the signed receipt. If the original SignedData object was sent encrypted within an EnvelopedData object and the ESS signed receipt is sent unencrypted, then the message digest value calculated for the original encrypted SignedData object is sent unencrypted. The responder should consider this when deciding whether or not to encrypt the ESS signed receipt.

2.4.1 MLExpansionHistory Attributes and Receipts

An MLExpansionHistory attribute MUST NOT be included in the attributes of a SignerInfo in a SignedData object that encapsulates a Receipt content. This is true because when a SignedData/Receipt is sent to an MLA for distribution, then the MLA MUST always encapsulate the received SignedData/Receipt in an outer SignedData in which the MLA will include the MLExpansionHistory attribute. The MLA cannot change the signedAttrs of the received SignedData/Receipt object, so it can't add the MLExpansionHistory to the SignedData/Receipt.

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. The receiptsTo field must be present in the receiptRequest

attribute. The software initiates the sequence of recipients with the value(s) of receiptsTo.

2. If the receiptPolicy attribute is present in the outer SignedData block and contains a value of insteadOf, then the software

replaces the sequence of recipients with the value(s) of insteadOf.

3. If the receiptPolicy attribute is present in the outer SignedData block and contains a value of inAdditionTo, then the software adds the value(s) of inAdditionTo to the sequence of recipients.

2.6. Signed Receipt Validation

A signed receipt is communicated as a single ASN.1 encoded object composed of a SignedData object directly including a Receipt content. It is identified by the presence of the id-ct-receipt object identifier in the encapContentInfo eContentType value of the SignedData object including the Receipt content.

Although recipients are not supposed to send more than one signed receipt, receiving agents SHOULD be able to accept multiple signed receipts from a recipient.

A SignedData/Receipt is validated as follows:

1. ASN.1 decode the SignedData object including the Receipt content.

2. Extract the contentType, signedContentIdentifier, and originatorSignatureValue from the decoded Receipt structure to identify the original SignedData signerInfo that requested the SignedData/Receipt.

3. Acquire the message signature digest value calculated by the sender to generate the signature value included in the original SignedData signerInfo that requested the SignedData/Receipt.

   1.1. If the sender-calculated message signature digest value has been saved locally by the sender, it needs be located and retrieved.

   2.2. If it has not been saved, then it needs be re-calculated based on the original SignedData content and signedAttrs as described in [CMS].

4. The message signature digest value calculated by the sender is then compared with the value of the msgSigDigest signedAttribute

included in the SignedData/Receipt signerInfo. If these digest values are identical, then that proves that the message signature digest value calculated by the recipient based on the received original SignedData object is the same as that calculated by the sender. This proves that the recipient received exactly the same original SignedData content and signedAttrs as sent by the sender because that is the only way that the recipient could have calculated the same message signature digest value as calculated by the sender. If the digest values are different, then the SignedData/Receipt signature verification process fails.

7. Acquire the digest value calculated by the sender for the Receipt content constructed by the sender (including the contentType, signedContentIdentifier, and signature value that were included in the original SignedData signerInfo that requested the SignedData/Receipt).

   5.1. If the sender-calculated Receipt content digest value has been saved locally by the sender, it needs be located and retrieved.

   5.2. If it has not been saved, then it needs be re-calculated. As described in section above, step 2, create a Receipt structure including the contentType, signedContentIdentifier and signature value that were included in the original SignedData signerInfo that requested the signed receipt. The Receipt structure is then ASN.1 DER encoded to produce a data stream which is then digested to produce the Receipt content digest value.

6. The Receipt content digest value calculated by the sender is then compared with the value of the messageDigest signedAttribute included in the SignedData/Receipt signerInfo. If these digest values are identical, then that proves that the values included in the Receipt content by the recipient are identical to those that were included in the original SignedData signerInfo that requested the SignedData/Receipt. This proves that the recipient received the original SignedData signed by the sender, because that is the only way that the recipient could have obtained the original SignedData signerInfo signature value for inclusion in the Receipt content. If the digest values are different, then the SignedData/Receipt signature verification process fails.

7. The ASN.1 DER encoded signedAttrs of the SignedData/Receipt signerInfo are digested as described in [CMS].

8. The resulting digest value is then used to verify the signature value included in the SignedData/Receipt signerInfo. If the signature verification is successful, then that proves the integrity of the SignedData/receipt signerInfo signedAttrs and authenticates the identity of the signer of the SignedData/Receipt signerInfo. Note that the signedAttrs include the recipient-calculated Receipt content digest value (messageDigest attribute) and recipient-calculated message signature digest value (msgSigDigest attribute). Therefore, the aforementioned comparison of the sender-generated and recipient-generated digest values combined with the successful SignedData/Receipt signature verification proves that the recipient received the exact original SignedData content and signedAttrs (proven by msgSigDigest attribute) that were signed by the sender of the original SignedData object (proven by messageDigest attribute). If the signature verification fails, then the SignedData/Receipt signature verification process fails.

The signature verification process for each signature algorithm that is used in conjunction with the CMS protocol is specific to the algorithm. These processes are described in documents specific to the algorithms.

2.7 Receipt Request Syntax

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

```
ReceiptRequest ::= SEQUENCE {
  signedContentIdentifier ContentIdentifier,
  receiptsFrom ReceiptsFrom,
  receiptsTo SEQUENCE SIZE (1..ub-receiptsTo)) OF GeneralNames }

ub-receiptsTo INTEGER ::= 16

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

ContentIdentifier ::= OCTET STRING

id-aa-contentIdentifier OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 7}
```

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:

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

```
ReceiptsFrom ::= CHOICE {
  allOrFirstTier [0] AllOrFirstTier,
  -- formerly "allOrNone [0]AllOrNone"
  receiptList [1] SEQUENCE OF GeneralNames }

AllOrFirstTier ::= INTEGER { -- Formerly AllOrNone
  allReceipts (0),
  firstTierRecipients (1) }
```

The receiptsTo field is used by the originator to identify the user(s) to whom the identified recipient needs to send signed

receipts. The message originator MUST populate the receiptsTo field with a GeneralNames for each entity to whom the recipient is suppose to send the signed receipt. If the message originator wants the recipient to send the signed receipt to the originator, then the originator MUST include a GeneralNames for itself in the receiptsTo field.

2.8 Receipt Policy Syntax

Various entities can modify how receipt processing is done; this is accomplished by adding a receiptPolicy attribute to a signature layer. A receiptPolicy attribute has an ASN.1 type of ReceiptPolicy. Use the receiptPolicy attribute only within the signed attributes associated with a signed message.

```
ReceiptPolicy ::= CHOICE {
  none [0] NULL,
  insteadOf [1] SEQUENCE SIZE (1..MAX) OF GeneralNames,
  inAdditionTo [2] SEQUENCE SIZE (1..MAX) OF GeneralNames }

id-aa-receiptPolicy OBJECT IDENTIFIER ::= {id-aa XX}
```

2.8.1 Receipt Policy Combining

There are circumstances where multiple receiptPolicy attributes need to be combined together. (One example is during MLA processing where multiple signature layers are removed.) This section gives the rules for combining two attributes. Attribute A is the inner of the two receiptPolicy attributes. The final result of combining two policies together should be the same as if the two policies were processed in sequence.

The following table describes the outcome of the union of ReceiptPolicy A (the rows in the table) and ReceiptPolicy B (the columns in the table).

```
             |                    B's policy
A's policy   | none   insteadOf      inAdditionTo
-------------------------------------------------
none         | none   none           none
insteadOf    | none   insteadOf(B)   *1
inAdditionTo | none   insteadOf(B)   *2
```

*1 = insteadOf(insteadOf(A) + inAdditionTo(B))
*2 = inAdditionTo(inAdditionTo(A) + inAdditionTo(B))

2.8 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 ESSVersion,
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

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

ESSVersion ::= INTEGER  { v1(1) }
```

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

2.9 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 UTF8String (SIZE (1..MAX)) OPTIONAL,
  contentType ContentType }

id-aa-contentHint OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 4}
```

The contentDescription field is used to provide information that the recipient can use to select protected messages for processing, such as a message subject. If this field is set, then the attribute is expected to appear on the SignedData object enclosing an EnvelopedData object and not on the inner SignedData object. The (SIZE (1..MAX)) construct constrains the sequence to have at least one entry. MAX indicates the upper bound is unspecified. Implementations are free to choose an upper bound that suits their environment.

Messages that contain a SignedData object wrapped around an EnvelopedData object, thus masking the inner content type of the message, SHOULD include a contentHints attribute, except for the case of the data content type. Specific message content types can either force or preclude the inclusion of the contentHints attribute. For example, when a SignedData/Receipt is encrypted within an EnvelopedData object, an outer SignedData object MUST be created that encapsulates the EnvelopedData object and a contentHints attribute with contentType set to the id-ct-receipt object identifier MUST be included in the outer SignedData SignerInfo signedAttrs.

2.10 Message Signature Digest Attribute

The msgSigDigest attribute can only be used in the signed attributes of a signed receipt. It contains the digest of the ASN.1 DER encoded signedAttrs included in the original SignedData that requested the signed receipt. Only one msgSigDigest attribute can appear in a signed attributes set. It is defined as follows:

```
msgSigDigest ::= OCTET STRING

id-aa-msgSigDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 5}
```

2.11 Signed Content Reference Attribute

The contentReference attribute is a link from one SignedData to another. It is used to link a reply to the original message to which it refers, or to incorporate by reference one SignedData into another. The first SignedData MUST include a contentIdentifier signed attribute, which SHOULD be constructed as specified in section 2.7. The second SignedData links to the first by including a ContentReference signed attribute containing the content type, content identifier, and signature value from the first SignedData.

```
ContentReference ::= SEQUENCE {
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

id-aa-contentReference   OBJECT IDENTIFIER ::= { iso(1) member-
body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2)
10 }
```

4. Mail List Management

Sending agents need to 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 succession of mail that will be distributed to all other members of both lists.

To prevent mail loops, MLAs use the mlaExpandHistory attribute of the outer signature of a triple wrapped message. The mlaExpandHistory 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 mlaExpandHistory attribute, located in the signed attributes of the MLA's SignerInfo block. The MLA creates or updates the signed mlaExpandHistory attribute value each time the MLA expands and signs a message for members of a mail list.

The MLA MUST add an MLAData record containing the MLA's identification information, date and time of expansion to the end of the mail list expansion history sequence. If the mlaExpandHistory attribute is absent, then the MLA MUST add the attribute and the current expansion becomes the first element of the sequence. If the mlaExpandHistory attribute is present, then the MLA MUST add the current expansion information to the end of the existing MLAExpandHistory sequence. Only one mlaExpandHistory attribute can be included in the signedAttrs of a SignerInfo.

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

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

If an MLA creates a SignerInfo that includes an mlaExpandHistory attribute, then all of the SignerInfos created by the MLA for that SignedData object MUST include an mlaExpandHistory attribute, and the value of each MUST be identical. Note that other agents might later add SignerInfo attributes to the SignedData block, and those additional SignerInfos might not include mlaExpandHistory attributes.

A recipient MUST verify the signature of the SignerInfo that covers the mlaExpandHistory attribute before processing the mlaExpandHistory, and MUST NOT process the mlaExpandHistory attribute unless the signature over it has been verified. If a SignedData object has more than one SignerInfo that has an mlaExpandHistory attribute, the recipient MUST compare the mlaExpandHistory attributes

in all the SignerInfos that it has verified, and MUST NOT process the mlaExpandHistory attribute unless every verified mlaExpandHistory attribute in the SignedData block is identical. If the mlaExpandHistory attributes in the verified signerInfos are not all identical, then the receiving agent MUST stop processing the message and SHOULD notify the user or MLA administrator of this error condition. In the mlaExpandHistory processing, SignerInfos that do not have an mlaExpandHistory attribute are ignored.

4.1.1 Detecting Mail List Expansion Loops

Prior to expanding a message, the MLA examines the value of any existing mlaExpandHistory 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 MLAData entry found in the mlaExpandHistory. 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

The first few paragraphs of this section provide a high-level description of MLA processing. The rest of the section provides a detailed description of MLA processing.

MLA message processing depends on the structure of the S/MIME layers in the message sent to the MLA for expansion. In addition to sending triple wrapped messages to an MLA, an entity can send other types of messages to an MLA, such as:

- a single wrapped SignedData or EnvelopedData message
- a double wrapped message (such as signed and enveloped, envelopedand signed, or signed and signed, and so on)
- a quadruple-wrapped message (such as if a well-formed triple wrapped message was sent through a gateway that added an outer SignedData layer)

In all cases, the MLA MUST parse all layers of the received message to determine if there are any SignedData layers that include an eSSSecurityLabel signedAttribute. This can include decrypting an EnvelopedData layer to determine if an encapsulated SignedData layer includes an eSSSecurityLabel attribute. The MLA MUST fully process each eSSSecurityLabel attribute found in the various SignedData layers, including performing access control checks, before distributing the message to the ML members. The details of the access

control checks are beyond the scope of this document. The MLA MUST verify the signature of the signerInfo including the eSSSecurityLabel attribute before using it.

In all cases, the MLA MUST sign the message to be sent to the ML members in a new "outer" SignedData layer. The MLA MUST add or update an mlaExpandHistory attribute in the "outer" SignedData that it creates to document MLA processing. If there was an "outer" SignedData layer included in the original message received by the MLA, then the MLA-created "outer" SignedData layer MUST include each signed attribute present in the original "outer" SignedData layer, unless the MLA explicitly replaces an attribute (such as signingTime or mlaExpandHistory) with a new value.

When an S/MIME message is received by the MLA, the MLA MUST first determine which received SignedData layer, if any, is the "outer" SignedData layer. To identify the received "outer" SignedData layer, the MLA MUST verify the signature and fully process the signedAttrs in each of the outer SignedData layers (working from the outside in) to determine if any of them either include an mlaExpandHistory attribute or encapsulate an EnvelopedData object.

The MLA's search for the "outer" SignedData layer is completed when it finds one of the following:

- the "outer" SignedData layer that includes an mlaExpandHistory attribute or encapsulates an EnvelopedData object
- an EnvelopedData layer
- the original content (that is, a layer that is neither EnvelopedData nor SignedData).

If the MLA finds an "outer" SignedData layer, then the MLA MUST perform the following steps:

1. Strip off all of the SignedData layers that encapsulated the "outer" SignedData layer

2. Strip off the "outer" SignedData layer itself (after remembering the included signedAttrs)

3. Expand the EnvelopedData (if present)

4. Sign the message to be sent to the ML members in a new "outer" SignedData layer that includes the signedAttrs (unless explicitly replaced) from the original, received "outer" SignedData layer.

If the MLA finds an "outer" SignedData layer that includes an

mlaExpandHistory attribute AND the MLA subsequently finds an EnvelopedData layer buried deeper with the layers of the received message, then the MLA MUST strip off all of the SignedData layers down to the EnvelopedData layer (including stripping off the original "outer" SignedData layer) and MUST sign the expanded EnvelopedData in a new "outer" SignedData layer that includes the signedAttrs (unless explicitly replaced) from the original, received "outer" SignedData layer.

If the MLA does not find an "outer" SignedData layer and does not find an EnvelopedData layer, then the MLA MUST sign the original, received message in a new "outer" SignedData layer. If the MLA does not find an "outer" SignedData and does find an EnvelopedData layer then it MUST expand the EnvelopedData layer, if present, and sign it in a new "outer" SignedData layer.

4.2.1 Examples of Rule Processing

The following examples help explain the rules above:

1) A message (S1(Original Content)) (where S = SignedData) is sent to the MLA in which the SignedData layer does not include an mlaExpandHistory attribute. The MLA verifies and fully processes the signedAttrs in S1. The MLA decides that there is not an original, received "outer" SignedData layer since it finds the original content, but never finds an EnvelopedData and never finds an mlaExpandHistory attribute. The MLA calculates a new SignedData layer, S2, resulting in the following message sent to the ML recipients: (S2(S1(Original Content))). The MLA includes an mlaExpandHistory attribute in S2.

2) A message (S3(S2(S1(Original Content)))) is sent to the MLA in which none of the SignedData layers includes an mlaExpandHistory attribute. The MLA verifies and fully processes the signedAttrs in S3, S2 and S1. The MLA decides that there is not an original, received "outer" SignedData layer since it finds the original content, but never finds an EnvelopedData and never finds an mlaExpandHistory attribute. The MLA calculates a new SignedData layer, S4, resulting in the following message sent to the ML recipients: (S4(S3(S2(S1(Original Content))))). The MLA includes an mlaExpandHistory attribute in S4.

3) A message (E1(S1(Original Content))) (where E = EnvelopedData) is sent to the MLA in which S1 does not include an MLAExpandHistory attribute. The MLA decides that there is not an original, received "outer" SignedData layer since it finds the E1 as the outer layer.

The MLA expands the recipientInformation in E1. The MLA calculates a new SignedData layer, S2, resulting in the following message sent to the ML recipients: (S2(E1(S1(Original Content)))). The MLA includes an mlaExpandHistory attribute in S2.

4) A message (S2(E1(S1(Original Content)))) is sent to the MLA in which S2 includes an mlaExpandHistory attribute. The MLA verifies the signature and fully processes the signedAttrs in S2. The MLA finds the mlaExpandHistory attribute in S2, so it decides that S2 is the "outer" SignedData. The MLA remembers the signedAttrs included in S2 for later inclusion in the new outer SignedData that it applies to the message. The MLA strips off S2. The MLA then expands the recipientInformation in E1 (this invalidates the signature in S2 which is why it was stripped). The MLA calculates a new SignedData layer, S3, resulting in the following message sent to the ML recipients: (S3(E1(S1(Original Content)))). The MLA

includes in S3 the attributes from S2 (unless it specifically replaces an attribute value) including an updated mlaExpandHistory attribute.

5) A message (S3(S2(E1(S1(Original Content))))) is sent to the MLA in which none of the SignedData layers include an mlaExpandHistory attribute. The MLA verifies the signature and fully processes the signedAttrs in S3 and S2. When the MLA encounters E1, then it decides that S2 is the "outer" SignedData since S2 encapsulates E1. The MLA remembers the signedAttrs included in S2 for later inclusion in the new outer SignedData that it applies to the message. The MLA strips off S3 and S2. The MLA then expands the recipientInformation in E1 (this invalidates the signatures in S3 and S2 which is why they were stripped). The MLA calculates a new SignedData layer, S4, resulting in the following message sent to the ML recipients: (S4(E1(S1(Original Content)))). The MLA includes in S4 the attributes from S2 (unless it specifically replaces an attribute value) and includes a new mlaExpandHistory attribute.

6) A message (S3(S2(E1(S1(Original Content))))) is sent to the MLA in which S3 includes an mlaExpandHistory attribute. In this case, the MLA verifies the signature and fully processes the signedAttrs in S3. The MLA finds the mlaExpandHistory in S3, so it decides that S3 is the "outer" SignedData. The MLA remembers the signedAttrs included in S3 for later inclusion in the new outer SignedData that it applies to the message. The MLA keeps on parsing encapsulated layers because it must determine if there are any eSSSecurityLabel attributes contained within. The MLA verifies the signature and fully processes the signedAttrs in S2. When the MLA

encounters E1, then it strips off S3 and S2. The MLA then expands the recipientInformation in E1 (this invalidates the signatures in S3 and S2 which is why they were stripped). The MLA calculates a new SignedData layer, S4, resulting in the following message sent to the ML recipients: (S4(E1(S1(Original Content)))). The MLA includes in S4 the attributes from S3 (unless it specifically replaces an attribute value) including an updated mlaExpandHistory attribute.

4.2.3 Processing Choices

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

4.2.3.1 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. The MLA also removes the existing originatorInfo field and replaces it with a new originatorInfo value built from information describing the MLA.

3. The MLA encapsulates the expanded encrypted message in a SignedData block, adding an mlExpandHistory 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.

4.2.3.2 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 CMS 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 a signed mlaExpandHistory attribute, the MLA checks for an expansion loop as described in the "Detecting Mail List Expansion Loops" section, then go to step 3. If the outermost SignedData layer does not include a signed mlaExpandHistory attribute, the MLA signs the whole message (including this outermost SignedData layer that doesn't have an mlaExpandHistory attribute), and delivers the updated message to mail list members to complete MLA processing.

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 mlaExpandHistory and all other signed attributes in that layer, if present.

   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 outermost SignedData layer created by the MLA replaces the original outermost SignedData layer. The MLA MUST create a signed attribute list for the new outermost SignedData layer which MUST include each signed attribute present in the original outermost SignedData layer, unless the MLA explicitly replaces one or more particular attributes with new value. A special case is the mlaExpandHistory attribute. The MLA MUST add an mlaExpandHistory signed

attribute to the outer SignedData layer as follows:

3.2.3.1. If the original outermost SignedData layer included an mlaExpandHistory 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 mlaExpandHistory attribute, a new attribute value is created with the current ML expansion information.

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

3.3.1. The MLA encapsulates the received SignedData object in an outer SignedData object, and adds an mlaExpandHistory 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 mlaExpandHistory?
       YES -> Check it, then -> 3.
       NO  -> Sign message (including outermost SignedData that
              doesn't have mlaExpandHistory), deliver it, STOP.
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
       mlaExpandHistory and other signed attributes, then -> 3.2.2.
3.2.2. Encapsulate in new signature, then -> 3.2.3.
3.2.3. Create new SignedData layer.
       Was there an old mlaExpandHistory?
       YES -> copy the old mlaExpandHistory values, then -> 4.
       NO  -> create new mlaExpandHistory value, then -> 4.
```

3.3. Encapsulate in a SignedData layer and add an mlaExpandHistory attribute, then -> 4.
4. Sign message, deliver it, STOP.

4.2.3.3 Processing for data

1. The MLA encapsulates the message in a SignedData layer, and adds an mlaExpandHistory 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 Agent Signed Receipt Policy Processing

If a mailing list (B) is a member of another mailing list (A), list B often needs to propagate forward the mailing list receipt policy of A. As a general rule, a mailing list should be conservative in propagating forward the mailing list receipt policy because the ultimate recipient need only process the last item in the ML expansion history. The MLA builds the expansion history to meet this requirement.

The following table describes the outcome of the union of mailing list A's policy (the rows in the table) and mailing list B's policy (the columns in the table).

	B's policy			
A's policy	none	insteadOf	inAdditionTo	missing
none	none	none	none	none
insteadOf	none	insteadOf(B)	*1	insteadOf(A)
inAdditionTo	none	insteadOf(B)	*2	inAdditionTo(A)
missing	none	insteadOf(B)	inAdditionTo(B)	missing

*1 = insteadOf(insteadOf(A) + inAdditionTo(B))
*2 = inAdditionTo(inAdditionTo(A) + inAdditionTo(B))

4.4 Mail List Expansion History Syntax

An mlaExpandHistory attribute value has ASN.1 type MLAExpandHistory. If there are more than ub-ml-expansion-history mailing lists in the sequence, the receiving agent should provide notification of the error to a human mail list administrator. The mail list administrator is responsible for correcting the overflow condition.

```
MLAExpandHistory ::= SEQUENCE
     SIZE (1..ub-ml-expansion-history) OF MLAData

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

ub-ml-expansion-history INTEGER ::= 64
```

MLAData 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 MLAData structure.

```
MLAData ::= SEQUENCE {
  mailListIdentifier EntityIdentifier,
  expansionTime GeneralizedTime }

EntityIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier SubjectKeyIdentifier }
```

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. In the event that a ML's signed receipt policy supersedes the originator's request for signed receipts, such that the originator will not receive any signed receipts, then the MLA MAY inform the originator of that fact.

A. ASN.1 Module

```
ExtendedSecurityServices2003
     { iso(1) member-body(2) us(840) rsadsi(113549)
       pkcs(1) pkcs-9(9) smime(16) modules(0) ess2003(XX) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS

-- Cryptographic Message Syntax (CMS)
    ContentType, IssuerAndSerialNumber, SubjectKeyIdentifier
    FROM CryptographicMessageSyntax { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cms(1)}

-- PKIX Certificate and CRL Profile, Sec A.2 Implicitly Tagged Module,
```

```
--  1988 Syntax
    PolicyInformation, CertificateSerialNumber, GeneralNames
    FROM PKIX1Implicit88 {iso(1)
    identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7)id-mod(0) id-pkix1-implicit(19)};

-- Extended Security Services

-- The construct "SEQUENCE SIZE (1..MAX) OF" appears in several ASN.1
-- constructs in this module. A valid ASN.1 SEQUENCE can have zero or
-- more entries. The SIZE (1..MAX) construct constrains the SEQUENCE to
-- have at least one entry. MAX indicates the upper bound is
unspecified.
-- Implementations are free to choose an upper bound that suits their
-- environment.

UTF8String ::= [UNIVERSAL 12] IMPLICIT OCTET STRING
    -- The contents are formatted as described in [UTF8]

-- Section 2.7

ReceiptRequest ::= SEQUENCE {
  signedContentIdentifier ContentIdentifier,
  receiptsFrom ReceiptsFrom,
  receiptsTo SEQUENCE SIZE (1..ub-receiptsTo) OF GeneralNames }

ub-receiptsTo INTEGER ::= 16

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

ContentIdentifier ::= OCTET STRING

id-aa-contentIdentifier OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 7}

ReceiptsFrom ::= CHOICE {
  allOrFirstTier [0] AllOrFirstTier,
  -- formerly "allOrNone [0]AllOrNone"
  receiptList [1] SEQUENCE OF GeneralNames }

AllOrFirstTier ::= INTEGER { -- Formerly AllOrNone
  allReceipts (0),
  firstTierRecipients (1) }

-- Section 2.X

id-aa-receiptPolicy ::= {id-at XX}

ReceiptPolicy ::= CHOICE {
  none [0] NULL,
  insteadOf [1] SEQUENCE SIZE (1..MAX) OF GeneralNames,
```

```
  inAdditionTo [2] SEQUENCE SIZE (1..MAX) OF GeneralNames }
```



```
-- Section 2.8

Receipt ::= SEQUENCE {
  version ESSVersion,
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

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

ESSVersion ::= INTEGER  { v1(1) }

-- Section 2.9

ContentHints ::= SEQUENCE {
  contentDescription UTF8String (SIZE (1..MAX)) OPTIONAL,
  contentType ContentType }

id-aa-contentHint OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 4}

-- Section 2.10

MsgSigDigest ::= OCTET STRING

id-aa-msgSigDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 5}

-- Section 2.11

ContentReference ::= SEQUENCE {
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

id-aa-contentReference   OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 10 }

-- Section 3.2

ESSSecurityLabel ::= SET {
  security-policy-identifier SecurityPolicyIdentifier,
```

```
  security-classification SecurityClassification OPTIONAL,
  privacy-mark ESSPrivacyMark OPTIONAL,
  security-categories SecurityCategories OPTIONAL }

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



```
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

ESSPrivacyMark ::= CHOICE {
    pString      PrintableString (SIZE (1..ub-privacy-mark-length)),
    utf8String   UTF8String (SIZE (1..MAX))
}

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 -- 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

-- Section 3.4

EquivalentLabels ::= SEQUENCE OF ESSSecurityLabel

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

-- Section 4.4

id-aa-mlaExpandHistory OBJECT IDENTIFIER ::= {id-aa X }
```



```
MLAExpandHistory ::= SEQUENCE
                SIZE (1..ub-ml-expansion-history) of MLAData

MLAData ::= SEQUENCE {
  mailListIdentifier EntryIdentifier,
  expansionTime GeneralizedTime
}


-- The use of id-aa-mlExpandHistory is obsoleted and replaced by
--    id-aa-mlaExpandHistory and id-aa-receiptBehavior

MLAExpandHistory ::= SEQUENCE
        SIZE (1..ub-ml-expansion-history) OF MLAData

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

ub-ml-expansion-history INTEGER ::= 64

MLAData ::= SEQUENCE {
  mailListIdentifier EntityIdentifier,
  expansionTime GeneralizedTime }

EntityIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier SubjectKeyIdentifier }

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

```
-- Section 5.4

SigningCertificate ::=  SEQUENCE {
    certs        SEQUENCE OF ESSCertID,
    policies     SEQUENCE OF PolicyInformation OPTIONAL
}

id-aa-signingCertificate OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 12 }

ESSCertID ::=  SEQUENCE {
     certHash                 Hash,
     issuerSerial             IssuerSerial OPTIONAL
}

Hash ::= OCTET STRING -- SHA1 hash of entire certificate

IssuerSerial ::= SEQUENCE {
```



```
     issuer                   GeneralNames,
     serialNumber             CertificateSerialNumber
   }
--
--  The following items are included for historical reasons.
--  See Appendix C of this document for processing.
--

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

id-aa-mlExpandHistory OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 3}

ub-ml-expansion-history INTEGER ::= 64

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

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

END -- of ExtendedSecurityServices

C. Processing for Obsolete Mail List Expansion Signed Attribute

One of the main changes between this document and it's predecessor is the decomposistion of the MLExpansionHistory attribute into the MLAExpandHistory and ReceiptPolicy attributes. The author does not currently know of any systems that generate the MLExpansionHistory attribute, however this section is provided for completeness.

When an implementation finds the old MLExpansionHistory attribute the following is suggested as the correct handling:

1. If there exists a MLAExpandHistory or ReceiptPolicy attribute, ignore the MLExpansionHistory attribute for processing, but place it into the new signature created.

2. Decompose the MLExpansionHistory attribute into a MLAExpandHistory attribute and ReceiptPolicy attribute as necessary. Place the current MLExpansionHistory attribute in all new signatures created.

D. Acknowledgments

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.

The first RFC version of this work was edited by Paul Hoffman who did remarkably well in keeping up with the arguments between John, myself and the others who contributed to this document.

Many other people have contributed hard work to this memo, including:

Andrew Farrell
Bancroft Scott
Bengt Ackzell
Bill Flanigan
Blake Ramsdell
Carlisle Adams
Darren Harter
David Kemp
Denis Pinkas
Francois Rousseau
Russ Housley

Scott Hollenbeck
Steve Dusse

Author's Addresses

Jim Schaad
Soaring Hawk Consulting
PO Box 675
Gold Bar, 98251

Email: jimsch@exmsft.com

Copyright Statement