Plasma Service CMS Processing
draft-schaad-plasma-cms-01

Abstract

Secure MIME (S/MIME) defined a method of placing security labels on a Cryptographic Message Syntax (CMS) object. These labels are placed as part of the data signed and validated by the parties. This means that the message content is visible to the recipient prior to the label enforcement. In [EPS-WS-TRUST] a new model has been presented where a third party is used as the enforcement point of the label. This document provides the details needed to implement the new Plasma model in the CMS infrastructure.

Additional benefits of using the Plasma module include moving responsibility of building lock boxes to the server and determining, based on policy, who should be a message recipient.

The document describes and details how the encryption process is performed, defines a new lock box attribute to hold the information needed to valid the label and to obtain the keys needed to decrypt the message. The document does not cover the protocol between the client and the Plasma policy enforcement server.

Status of this Memo

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

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

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

This Internet-Draft will expire on December 8, 2012.

Copyright Notice
1. Introduction

In the traditional S/MIME (Secure MIME) e-mail system, the sender of a message is responsible for determining the list of recipients for a message, obtaining a valid public or group key for each recipient, applying a security label to a message, and sending the message. The recipient of a message is responsible for the enforcement of any security labels on a message. While this system works in theory, in practice it has some difficulties that have lead to problems in getting S/MIME mail widely deployed. This document is part of an effort to provide an alternative method of allocating the responsibilities above to different entities in an attempt to make the process work better.

In an Email Policy Service (PLASMA) deployment of S/MIME, the sender of the message is still responsible for determining the list of recipients for the message and determining the security label to be applied to the message, however the responsibility of obtaining valid keys for each recipient can be off-loaded to the Plasma server component. The recipient is no longer responsible for enforcement of the policy as this is off-loaded to the Plasma server component. Doing this allows for the following changes in behavior of the system:

- The sender is no longer responsible for finding and validating the set of public keys used for the message. This simplifies the complexity of the sender and lowers the resources required by the sender. This is especially true when a large number of recipients is involved.

- The set of recipients that can decrypt the message can be change dynamically after the message is sent, without resorting to a group keying strategy.

- The enforcement of the policy is done centrally, this means that updates to the policy are instantaneous and the enforcement policy can be centrally audited.

- The label enforcement is done before the message is decrypted, this means there are no concerns about the message contents being leaked by poor client implementations.

- Many of the same components used in a web-based deployment of policy enforcement in a confederation can be used for e-mail based deployment of information. This includes using credentials other than X.509 certificates.

This document is laid out as follows:
o In Section 2 a more complete description of the components involved in the model are discussed.

o In Section 3 is description the new ASN.1 structures that are used to carry the additional information, and the way that these structures are used in a recipient info structure.

o In Section 4 is a description of the modifications from the sender processing rules outlined in [SMIME-MSG].

o In Section 5 is a description of the modification from the recipient processing rules outlined in [SMIME-MSG].

1.1. Vocabulary

Some of the terms used in this document include:

Authenticated Encryption with Additional Data (AEAD): Are a set of encryption algorithms where an authentication method stronger than the PKCS #1 packing method is used for authentication and, optionally, a set of unencrypted attribute values are included in the authentication step.

Content Encryption Key (CEK): The symmetric key used to encryption the content of a message.

Key Encryption Key (KEK): The encryption of a CEK by another key. This may be done by either a symmetric key or an asymmetric key.

1.2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
2. Model

To be supplied from the problem statement document.

2.1. Overview

To be supplied from the problem statement document.

List the boxes above and give some info about them.

2.2. Sender

The general steps that need to be taken by the sender of an EPS message are listed below. The steps refer to the numbers in the upper half of Figure 1. Talk about the expansion in section x.x

1. This list needs to be re-done - it does not include early-binding issues.

2. The Sending Agent composes the message, determines the set of recipients and a policy label to be applied to the message.

3. The Sending Agent randomly creates a KEK.

4. The Sending Agent transmits the KEK, the list of recipients and the policy label to the Email Policy Service. One possible protocol for this is [EPS-WS-TRUST].

5. The Email Policy Service validates that the set of recipients, the sender and policy label are consistent.
6. The Email Policy Service returns an EPS-KEK attribute to the Sending Agent.

7. The Sending Agent creates a normal S/MIME encrypted data message, one of the recipient info structures being a KEK recipient using the KEK created in step 2 and the EPS-KEK attribute from step 5.

8. The Sending Agent send the message to the Mail Transfer Agent using SMTP or a similar protocol.

2.3. Recipient

The general steps that need to be taken by a Receiving Agent for an EPS messaging system. The steps refer to the bottom half of Figure 1. An expansion of this is covered in Section 5. [anchor6]

1. The Receiving Agent obtains the message from a Mail Transfer Agent using IMAP, POP or similar protocol.

2. The Receiving Agent recognizes that a KEK recipient info with an EPS-KEK attribute exists and validates the attribute.

3. The Receiving Agent sends the KEK identifier and the EPS-KEK attribute along with other information to the Email Policy Service.

4. The Email Policy Service evaluates the policy label and the recipient information.

5. The Email Policy Service returns the KEK to the Receiving Agent.

6. The Receiving Agent decrypts the message and displays it to the user.
3. Recipient Info Encoding

This documents defines a new Other Key Attribute to be used in the KEK Recipient Info structure. There are two distinct ways that the problem of defining a new recipient info structure for Plasma could be approached. The first would be to define a new recipient info structure to be placed in the Other Recipient Info structure. The second option is to create a new key attribute to be placed in the KEK Recipient Info structure.

The use of a new recipient info structure would have been the easiest to document and implement, if most major CMS implementations had kept up with the latest versions. However it is known that several implementations stopped with RFC 2630 [RFC2630] and it was not until RFC 3369 [RFC3369] that the other recipient info choice was introduced along with the language stating that implementations need to gracefully handle unimplemented alternatives in the choice. This means that if a new recipient info structure was defined and adopted, the mail message would fail decoding for many recipients, even for those recipients that had a key transfer or key agreement recipient info structure. For this reason it was decided that the second option would be chosen.

The use of the KEKRecipientInfo type may seem to be a stretch at first, it was defined for those situations where a symmetric key had already been distributed and either a specific key or a specific transformation on the key was to be applied in order to decrypt the KEK value. However, in the most generic situation, Plasma will distribute a symmetric key back to the client so the difference between the original usage and how Plasma uses it is when the symmetric key is distributed to the recipient. Additionally, it is easy for client implementations to make the determination of a Plasma recipient info by looking at the OID for the other key attribute structure.

The attribute is created only by a Plasma server and not by the client software. A protocol such as the one in RFC TBD1 [EPS-WS-TRUST] is used for clients to obtain the attribute from a Plasma server.

For the convenience of the reader we include the KEKRecipientInfo structure pieces here (copied from [CMS-ASN]:

---

Schaad Expires December 8, 2012 [Page 7]
KEKRecipientInfo ::= SEQUENCE {
  version CMSVersion, -- always set to 4
  kekid KEKIdentifier,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey }

KEKIdentifier ::= SEQUENCE {
  keyIdentifier OCTET STRING,
  date GeneralizedTime OPTIONAL,
  other OtherKeyAttribute OPTIONAL }

OtherKeyAttribute ::= SEQUENCE {
  keyAttrId  KEY-ATTRIBUTE.
     &id({SupportedKeyAttributes}),
  keyAttr    KEY-ATTRIBUTE.
     &Type({SupportedKeyAttributes}{@keyAttrId})
}

When you look at the KEKRecipientInfo structure you fill out the fields as follows:

version  is set to the value of 4.

kekid  is a sequence where the fields are:

  keyIdentifier  is a randomly generated value. The value is created without any internal semantics and should be unique within the message. It is suggested that the value be between 20 and 30 octets in length. The key identifier allows for a correlation to exist between keys returned from the Plasma server and specific KEKRecipientInfo structures.

date  is not used and is omitted.

other  is a sequence where the fields are:

  keyAttrId  contains the value id-keyatt-eps-token.

  keyAttr  contains a the value of the attribute. The details of this structure are covered in Section 3.1.

keyEncryptionAlgorithm  contains the identifier and the type information for the key encryption algorithm. The mandatory to implement algorithms are specified in section Section 7. This algorithm must be understandable by the sender of the message and by the recipient of the message, but it is not a requirement that the Plasma Server can process the algorithm.

encryptedKey contains the CEK encrypted by the KEK.

3.1. EPS Other Key Attribute

The EPS Other Key Attribute functions as the lock box for the KEK used in encrypting the CEK. In addition to the KEK, the lock box also contains the information that is needed by the Email Policy Server to know the policy(s) applied to the encrypted data and possible parameters for the policy and for the client to validate that the lock box applies to the encrypted content.

The relevant section from the ASN.1 module which contains the content is:

```
--
-- New Other Key Attribute value for Plasma
-- This structure holds the encrypted KEK value for the server
-- and other signed attributes used by the client for checking
-- the structure applies in this case
--
keyatt-eps-kek KEY-ATTRIBUTE ::= {
   SignedData IDENTIFIED BY id-keyatt-eps-token
}

id-keyatt-eps-token OBJECT IDENTIFIER ::= {iso(1) member-body(2)
   us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD2 }
```

We define a new KEY-ATTRIBUTE called keyatt-eps-kek. This attribute is identified by the id-keyatt-eps-token. The data structure that is associated with this key attribute is the CMS SignedData structure. The CMS SignedData structure is used directly without a CMS ContentInfo structure wrapping it.

The SignedData structure fields are filled as follows (some less significant fields are omitted):

encapContentInfo is a structure containing the fields:

- eContentType is id-envelopedData or id-ct-authEnvelopedData.
- eContent is CMS EnvelopedData or AuthEnvelopedData structure with the following fields:

  recipientInfos contains the lock box(s) for the Plasma servers(s) to get access to the encrypted data. There MUST NOT be recipient info structures added for any entity not trusted to correctly perform the policy decision processing.
See below for some additional discussion on what lock boxes need to be created.

encryptedContentInfo/authEncryptedContentInfo is a structure containing the following elements:

contentType is id-ct-eps-LockBox.

cryptEncryptionAlgorithm contains the identifier and parameters for the content encryption algorithm. This algorithm only needs to be understood by the Plasma service.

encryptedContent contains the encrypted EPS LockBox content. Details on this type are in the next section.

certificates SHOULD contain the set of certificates (up to but not including the trust anchor) needed to validate the set of signer info structures.

signerInfos will contain one or more signer info structures. In each signature the signed attributes:

* MUST contain the signing time, the message digest, the content type and the EPS url attributes.

* MAY contain the ESS security label attribute.

* other attributes can also be included.

When creating the recipient info structures for the EnvelopedData structure, there will normally only need to be a single entry in the sequence as the only entity that needs to decrypt the EPS Lockbox is the Email Policy Service. In the event that the service is distributed over multiple servers then multiple lock boxes may need to be created. One of the implications of the fact that the originator of the message is the only recipient is that, although the signing key needs to be contained in a certificate, there is no corresponding requirement that the encryption key needs to be in a certificate. Instead of using a certificate, a subject key identifier that is meaningful only to the Email Policy Service can be used.

3.2. EPS Content Type

The inner content type for an EPS Other Key Attribute is an EPS Lockbox. This content is contained in an encrypted CMS object which is encrypted by and for the Plasma server itself, as such the
contents and structure is known just to the Plasma server.

The relevant section from the ASN.1 module which defines this content is:

```plaintext
ct-eps-LockBox CONTENT-TYPE ::= {
  TYPE EPS-LockBox
  IDENTIFIED BY id-ct-eps-LockBox
}

id-ct-eps-LockBox OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) TBD1}

EPS-LockBox ::= SEQUENCE {
  label        OneLabel,
  keyList      KeyList,
  attrList     AttributeList OPTIONAL
}

KeyList ::= SEQUENCE {
  namedRecipients [0] SEQUENCE SIZE (1..MAX) OF NamedRecipient OPTIONAL,
  defaultRecipients [1] SEQUENCE SIZE (1..MAX) OF OneKek OPTIONAL
}(WITH COMPONENTS {
  ...
  namedRecipients    PRESENT
} |
 WITH COMPONENTS {
  ...
  defaultRecipients  PRESENT
})

NamedRecipient ::= SEQUENCE {
  recipient    IA5String, -- name of the recipient
  keyValue     SEQUENCE {
    kekIdentifier          OCTET STRING,
    keyValue               RecipientInfo
  }
}

OneKek ::= SEQUENCE {
  kekIdentifier          OCTET STRING,
```
In the above ASN.1, the following items are defined:

ct-eps-LockBox is a new CMS content type object, this object is added to the set of content type objects in ContentSet (defined in the ASN.1 module in [CMS-ASN]). The content type associates the object identifier id-ct-eps-LockBox with the data type EPS-LockBox.

id-ct-eps-LockBox is the identifier defined for the new content type.

EPS-LockBox is the new type defined for new content type. This is a sequence [anchor8] with the following fields:

label contains the policy label that is to be applied to the KEK values in the keyList field.

keyList contains the KEK values.

attrList contains a set of attributes which are considered as significant by the Plasma server internally.
KeyList is a new type that contains the KEK values or the KeyRecipientInfo structures. This allows for messages to be sent with either early-binding, where a RecipientInfo structure is filled out for the receiving agent, or late-binding, where the KEK value is sent from the Plasma Service to the receiving agent. It is required that at least one of these fields is populated.

namedRecipients contains the recipient info structures for individually identified recipients.[anchor9]

defaultRecipients contains the KEK keys for those recipients that are not individual identified with their own recipient info structures.

NamedRecipient contains the information identifying a single named recipient along with the recipient info structures for that recipient.

recipient contains the name of the name of the recipient in the form of an RFC2822 email address.

kekValue contains the recipient info structure for the named recipient. The fields in the sequence are:

kekIdentifier contains the value of the kek identifier in the message.

keyValue contains a recipient info structure wrapping the CEK of the message.

OneKek contains the information that defines a single KEK to be used. The sequence has the fields:

kekIdentifier contains the identification value for the KEK. This value matches the KEKIdentifier.keyIdentifier value in the recipient info information.

kekValue contains the KEK value.

OneLabel is the type structure used to specify the individual policies and how the policies interact with each other. The structure is explicitly setup to be extensible in future versions. Information on how the extensibility should be handled is in Section 3.2.1. The choices in the structure are:
andLabels allows for a set of policies to be combined together in such a way as to state that for the overall statement to be true, each of the policies listed must also be true. The ASN.1 is designed so that there must be at least two elements in the combined statement.

orLabels allows for a set of policies to be combined together in such a way as to state that for the overall statement to be true, any of the policies listed needs to be true. The ASN.1 is designed so that there must be at least two elements in the combined statement.

exclude allows for two policies to be combined together such as to state that for the overall policy to be true, the first policy must be true and the second policy must not be true. This policy combination is designed to remove a set of people from the overall policy. (I.e. every one in the general group but is not working for company X.)

uriLeaf allows for the specification of a policy as a URI. If the policy is unknown then the policy evaluation fails. The use of a URI allows for the addition of parameters to be added to the policy statement. [anchor10]

oidLeaf allows for the specification of a policy as an object identifier. There is no option to provide for parameters. [anchor11]

AttributeList defines a structure where a set of attributes can be included.

EPSAttributes defines an Object Set of attributes which can be included. The object set is intentionally open ended for later expansion. We currently do not define any items that go in this field.

3.2.1. Label Extensibility

The ASN.1 type OneLabel has been explicitly defined to allow for later extensibility. When a new element is added, it will be added with at the end of the choice with a different tag value. ASN.1 decoders need to following the current recommendations on dealing with extensibility. This means that when the decoder finds a new choice in this structure that is not part of the current syntax, the element should be treated as an unknown blob and returned to the caller as an unrecognized blob. This allows for the calling code to make the decision on how the unknown element is treated.
However the extensibility is not handled the same in all cases. Each of the four different cases is outlined below.

3.2.1.1. Sender Composing

The set of policies that can be used by the sender in applying a label is usually given as a list of policies, however under some circumstances the sender may be handed structured policies either for application or for checking that some policies can be used together. If structured policies are provided to the sender, it will not matter to the sender that they cannot evaluate the policy unless the details are to be shown to the sending user. Following the current ASN.1 rules which allow for the decoding and then re-encoding of a type which contains unknown nodes allows the sending agent the most flexibility.

The protocol used to give the policy information to the sending client may not use the ASN.1 structure provided here or configuration purposes but would generally be expected to provide for a different data format.

3.2.1.2. Sender to Email Policy Service

In the sending agent to Email Policy Service protocol (defined external to this document) the ASN.1 type OneLabel may or may not be used directly. If it is used, then the Email Policy Server is expected to reject the label if it contains elements which it does not understand. The general expectation is that the Email Policy Service that the sender is communicating with is going to be the same one as is later enforcing the policy. It makes no sense for this server to accept a policy that it would later be unable to enforce. The protocol should make provisions for the return of this as an explicit error code. Having the ASN.1 decoded allows for the communication of the exact tag that is causing the problem. Under most policies, the evaluation of sender policy would be expected to be different than laid out in the next session. As a general rule, the sender should be permitted to assert all of the leaf policies for the purpose of sending.

3.2.1.3. Recipient to Email Policy Service

The Email Policy Service which recipient communicates way is normally the same server as the sender communicated with. However the server can be a different server, or the server may have been downgraded in the mean time. In this case the policy evaluation need to be conservative. There are two different ways of doing this evaluation. The first option is to say that if an unknown node is found, then the policy evaluation results in "Deny" for all cases. The second option
is to try and evaluation the policy, but to do so in a conservative manner. In this section we use the same terms as XACML [XACML] uses in section 7.2.1. If the second option is chosen then the following rules are used:

uriLeaf results in "Permit", "Deny", "Not Applicable" or "Indeterminate". "Not Applicable" results if the policy is unknown. "Indeterminate" results if the policy cannot be evaluated.

oidLeaf results in "Permit", "Deny", "Not Applicable" or "Indeterminate". "Not Applicable" results if the policy is unknown. "Indeterminate" results if the policy cannot be evaluated.

andLabels results in "Deny" if any input node is "Deny" or "Not Applicable". It results in "Permit" if all of the input nodes are "Permit". Otherwise it results in "Indeterminate".

orLabels results in "Permit" if any input node is "Permit". It results in "Deny" if all nodes are either "Deny" or "Not Applicable". Otherwise it results in "Indeterminate".

exclude results in "Deny" if the first element is "Deny" or if the second input is "Permit". It results in "Permit" if the first input is "Permit" and the second is "Deny". It results in "Not Applicable" if either element is "Not Applicable". Otherwise it results in "Indeterminate".

If the final node results in "Permit", then access is granted. If the final result is either "Deny" or "Not Applicable" then access is denied. If the final result is "Indeterminate", then access is denied, however if the protocol permits, then the result can be "Not Applicable" and the attributes needed to do the policy evaluation can be requested and policy evaluation may be re-attempted.

Any future element that is added to the choice needs to define a similar rule to the set of rules above.

3.2.1.4. Recipient User Agent Display

Recipient user agents may want to display the policy to the user. This policy may be communicated from the Email Policy Service to the recipient using the OneLabel ASN.1 structure or using a different type. The label has been successfully (or unsuccessfully) validated so access has been granted (or denied) to the message. At this point we are only talking about a user interface issue. The recipient user agent should make some type of provision for indicating that an
operation was placed in that location of the tree, but the agent is not aware of what the operation is.

3.3. EPS URL Authenticated Attribute

It is required that the name of the Email Policy Server be securely communicated to the message recipient. For this purpose a URL is used as this can communicate both a server name as well as additional parameters that can be used to identify a specific service on the server.

The relevant section from the ASN.1 module for this attribute is:

```
-- Define the Signed Attribute to carry the
--       Email Policy Server URL
--
-- This attribute is added to the SignedAttributSet set of
-- attributes in [CMS-ASN]
--

aa-eps-url ATTRIBUTE ::= {
    TYPE UTF8String IDENTIFIED BY id-aa-eps-url
}

id-aa-eps-url OBJECT IDENTIFIER ::= { iso(1) member-body(2)
                                         us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD3}
```

From this we can see the following:

A new attribute aa-eps-url has been defined.

The OID value of id-aa-eps-url has been created to identify the new attribute.

The type of the value associated with the attribute is a UTF8String which contains the URL for the Email Policy Server. The URL defines both the destination server and the protocol to be used. When the schema for the URL is "https", then the protocol used is [eps-token]

The new attribute is to appear only as a Signed Attribute in a SignedData structure. It is therefore to be added to the attribute set SignedAttributeSet in the update ASN.1 module contained in [CMS-ASN].

The attribute structure defined for signed attributes allows for multiple values to be carried in a single attribute. For this
attribute there MUST be at least one value. If there is more than
one value, each value MUST be a unique. Multiple values are allowed
as there can be multiple Plasma servers that can be used to evaluate
the policy. The order of URLs does not indicate any order of
priority, any of the values may be used.

This attribute is only included in a SignedData object by an Email
Policy Server. There are no processing rules for the sender of a
message. The processing rules for a recipient can be found in
Section 5.

3.4. EPS Encrypted Content Hash Attribute

For privacy reasons, it is highly desirable that the recipient of a
message can validate that the Plasma lock box embedded in a message
is associated with encrypted data it is attached to. For this
reason, in addition to the requirement that a recipient validate the
signature of the Plasma server over the lock box, a new attribute is
defined which contains the hash of the encrypted content.

--
-- Define the Signed Attribute to carry the
--       hash of encrypted data
--
-- This attribute is added to the SignedAttributeSet set of
-- attributes in [CMS-ASN]
--

aa-eps-hash ATTRIBUTE ::= {
    TYPE HashValue IDENTIFIED BY id-aa-eps-hash
}

id-aa-eps-hash OBJECT IDENTIFIER ::= {iso(1) member-body(2)
   us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD4}

HashValue ::= SEQUENCE {
    hashAlgorithm    DigestAlgorithmIdentifier,
    hashValue        OCTET STRING
}

The above ASN.1 fragment defines the following items:

aa-eps-hash defines a new ATTRIBUTE object describing the encrypted
content hash attribute. This attribute is always a signed object
and is to be added to the SignedAttributeSet in the CMS ASN.1
module contained in [CMS-ASN].
id-aa-eps-hash defines the unique identifier of the attribute.

HashValue defines the data value to be associated with the attribute. The fields of this type are:

hashAlgorithm contains the identifier and parameters of the hash function used.

hashValue contains the value of the hash operation.

The hash is computed over the encrypted content, without including any of the ASN.1 wrapping around the content. Thus this value does not cover the content type identifier, the encryption algorithm and parameters or any authenticated attributes for AEAD algorithms.
4. Sender Processing Rules

4.1. Flow

This is the set of processing steps that a sender needs to do (the order of the steps is not normative):

1. Sender Agent obtains the set of policies under which it can send a message.
2. Sender Agent composes the message content.
3. Sender Agent determines the policy label to be applied to the message.
4. Sender Agent determines the set of recipients for the message.
5. Sender Agent randomly creates the KEK.
6. Sender Agent randomly selects the content encryption algorithm (with input from the policies chosen) and randomly creates the CEK.
7. Sender Agent encrypts the content with the CEK and computes the encrypted hash value.
8. Sender Agent may optionally create lock boxes for one or more of the message recipients, for those recipients which are to be protected by the policy server.
9. Sender Agent transmits the KEK, the list of recipients, the set of recipient lock boxes, the encrypted hash value and the policy label to the EPS.
10. Sender Agent receives an EPS-KEK attribute from the policy server. If the policy validation fails then the sender agent cannot send the message under the current policy label.
11. Sender Agent verifies the signature on the signed data structure inside of the EPS-KEK attribute.
   A. Signature is current and passes cryptographic processing.
   B. Signed attributes contains the EPS URL attribute, the EPS Encrypted Hash attribute and the attribute is consistent with the policy selected.
C. Other standard signature checks.

12. Sender Agent selects an appropriate content encryption algorithm and randomly generates a CEK and encrypts the message.

13. Sender Agent creates a KEK recipient info structure with the EPS-KEK attribute. Sender Agent also creates all other necessary recipient info structures (for recipients not protected by policy) including one itself if required.

14. Sender Agent finishes encoding the message and transmits it to the MTA.
5. Recipient Processing Rules

5.1. Flow

In this section we expand on the list of actions that are Section 2.3. When looking at the validation steps that are given here, the results need to be the same but the order that the steps are taken can be different. As an example, it can make sense to do the policy check in step X before doing the signature validation as this would not require any network access.

This is the set of processing that the recipient needs to do:

1. The Receiving Agent obtains the message from a Mail Transfer Agent using IMAP, POP or a similar protocol.

2. The Receiving Agent recognizes that a KEK recipient info exists with an EPS-KEK attribute. It is recommended that the entire list of recipient info structures be checked for one that can be processed directly before processing this node.

3. The Receiving Agent validates the EPS-KEK attribute. The following steps need to be taken for validation.

   A. The signature on the SignedData structure is validated. If the validation fails then processing ends. If more than one SignerInfo element exists on the structure, then the validation succeeds and the signed attributes from that SignerInfo structure are used. The order of performing the validation of the SignerInfo structures may be influenced by the content of EPS URL attribute.

   B. If an ESS security label attribute ([ESS-BASE]) is present, then it must be evaluated and processing ends if the security label processing fails or is denied.

   C. The EPS URL attribute is absent, then processing fails.

   D. The URL value in the EPS URL attribute is checked against local policy. If the check fails then processing fails. This check is performed so that information about the user is not given to random Email Policy Services.

   E. The EPS Encrypted Hash attribute value is checked against the encrypted content. If this check fails then processing fails.
4. The recipient gathers the necessary identity and attribute statements, usual certificates or SASL statements.

5. The recipient establishing a secure connection to the Email Policy server and passes in the identity and attribute statements and receives back the KEK or lock box to allow it to obtain the CEK value.

5.2. Reply and Forward Processing

In some circumstances a message recipient may be permitted to read a message sent under a certain policy, but it not permitted to send a message for that policy. In the event that a complex policy is used the recipient may be permitted to read under one policy, but not have any rights under a second policy. In both of these case a recipient of a message would be unable to either reply or forward a message using the same policy as they received it under. For this reason, the protocol used to communicate with the Plasma server will frequently return a single purpose policy that permits a recipient to reply to a message using the same policy as the original message.
6. S/MIME Capability

The SMIMECapabilities attribute was defined by S/MIME in [SMIME-MSG] so that the abilities of a client can be advertised to the recipients of an S/MIME message. This information can be advertised either directly in an S/MIME message sent from a client to a recipient, or more indirectly by publishing the information in an LDAP directory [RFC4262].

A new S/MIME capability is defined by this document so that a client can advertise to others that it understands how to deal with Plasma recipient information. The ASN.1 for this attribute is:

```
--
-- Create an S/MIME capability for advertising that
da client can understand the PLASMA recipient info
-- structure information
--
cap-Plasma-RecipientInfo SMIME-CAPS ::= {
  IDENTIFIED BY id-cap-plasma-recipientInfo
}

id-cap-plasma-recipientInfo OBJECT IDENTIFIER ::= {
id-cap TBD5
}
```

We define a new SMIME-CAPS object called cap-Plasma-RecipientInfo. This attribute is identified by the OID id-cap-plasma-recipientInfo and has no data structure associated with it. When encoded as an S/MIME capability the parameters MUST to be absent and not NULL.
7. Mandatory Algorithms

KEK mandatory to implement algorithms - MUST AES-128 KEK Wrap.
SHOULD AES-256 KEK wrap.

Key Transport - MUST RSA v 1.5

Key Agreement - MUST EC-DH for group ...

Content Encryption - MUST AES-128.
8. Security Considerations

Man in the middle attack on the protocol from the sending agent to the email policy server.

Man in the middle attack on the protocol from the receiving agent to the email policy server.
9. IANA Considerations

All of the object identifiers defined by this document are done so under the existing S/MIME Object Identifier arc. No actions by IANA are required for this document.
10. References

10.1. Normative References


10.2. Informative References


Editorial Comments

[anchor6]  Trevor: We need to clarify that you can still use the existing recipient info structure for backward compatibility. When that is appropriate is a matter of local policy.

[anchor8]  JLS: OPEN ISSUE: At one point there was a discussion that we would allow for multiple KEKs to be wrapped in a single object, you would then be able to retrieve server KEK values with the same label or with different labels at one shot. This would allow for multiple encrypted parts in a single message to have their keys wrapped and retrieved in one shot.

[anchor9]  trevor: OPEN ISSUE: Should we be using GeneralName so as to allow for name types other than email addresses.

[anchor10]  JLS: Do we want to change the type? What do we want to say about internationalization?

[anchor11]  JLS: We could add such an option if we desired.
Appendix A.  2009 ASN.1 Module

PolicySMime -- TBD Get a module number --
DEFINITIONS IMPLICIT TAGS ::= BEGIN
  IMPORTS
    -- Cryptographic Message Syntax (CMS) [RFC5652]
    CONTENT-TYPE, RecipientInfo, KEY-ATTRIBUTE, SignedData,
    DigestAlgorithmIdentifier
    FROM CryptographicMessageSyntax-2010
    { iso(1) member-body(2) us(840) rsadsi(113549)
      pkcs(1) pkcs-9(9) smime(16) modules(0) id-mod-cms-2009(58) }

    -- Common PKIX structures [RFC5912]
    SMIME-CAPS
    FROM AlgorithmInformation-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-algorithmInformation-02(58) }

    ATTRIBUTE, SingleAttribute()
    FROM PKIX-CommonTypes-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkixCommon-02(57) }

    ESSSecurityLabel
    FROM ExtendedSecurityServices-2009
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-ess-2006-02(42) }

    id-cap
    FROM SecureMimeMessage
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-msg-v3dot1-02(39) }

    ;
    --
    -- EPS Content Type
    --

    ct-eps-LockBox CONTENT-TYPE ::= {
      TYPE EPS-LockBox
      IDENTIFIED BY id-ct-eps-LockBox
    }
id-ct-eps-LockBox OBJECT IDENTIFIER ::= {iso(1) member-body(2)
  us(840) rsadsi(113549) pkcs(1) pkcs7(7) TBD1}

EPS-LockBox ::= SEQUENCE {
  label        OneLabel,
  keyList      KeyList,
  attrList     AttributeList OPTIONAL
}

KeyList ::= SEQUENCE {
  namedRecipients [0] SEQUENCE SIZE (1..MAX) OF 
    NamedRecipient OPTIONAL,
  defaultRecipients [1] SEQUENCE SIZE (1..MAX) OF 
    OneKek OPTIONAL
}

(WITH COMPONENTS {
  ...
  namedRecipients  PRESENT
}
| 
WITH COMPONENTS {
  ...
  defaultRecipients  PRESENT
})

NamedRecipient ::= SEQUENCE {
  recipient    IA5String, -- name of the recipient
  kekValue     SEQUENCE {
    kekIdentifier          OCTET STRING,
    keyValue               RecipientInfo
  }
}

OneKek ::= SEQUENCE {
  kekIdentifier       OCTET STRING,
  kekValue            OCTET STRING
}

OneLabel ::= CHOICE {
  andLabels     [1] SEQUENCE SIZE (2..MAX) OF OneLabel,
  orLabels      [2] SEQUENCE SIZE (2..MAX) OF OneLabel,
  exclude       [3] SEQUENCE SIZE (2) OF OneLabel,
  uriLeaf       [4] SEQUENCE {
    policyId        UTF8String,
    names           SEQUENCE SIZE (1..MAX) OF 
      IA5String OPTIONAL
  },
  oidLeaf       [5] ESSSecurityLabel,
  ...

AttributeList ::= SEQUENCE SIZE (1..MAX) OF SingleAttribute{{EPSAttributes}}

EPSAttributes ATTRIBUTE ::= { ... }

--
-- New Other Key Attribute value for Plasma
-- This structure holds the encrypted KEK value for the server
-- and other signed attributes used by the client for checking
-- the structure applies in this case
--
keyatt-eps-kek KEY-ATTRIBUTE ::= {
        SignedData IDENTIFIED BY id-keyatt-eps-token
    }

id-keyatt-eps-token OBJECT IDENTIFIER ::= {iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD2}

--
-- Define the Signed Attribute to carry the
-- Email Policy Server URL
-- This attribute is added to the SignedAttributSet set of
-- attributes in [CMS-ASN]
--

aa-eps-url ATTRIBUTE ::= {
    TYPE UTF8String IDENTIFIED BY id-aa-eps-url
}

id-aa-eps-url OBJECT IDENTIFIER ::= {iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD3}

--
-- Define the Signed Attribute to carry the
-- hash of encrypted data
-- This attribute is added to the SignedAttributeSet set of
-- attributes in [CMS-ASN]
--

aa-eps-hash ATTRIBUTE ::= {
    TYPE HashValue IDENTIFIED BY id-aa-eps-hash
}
id-aa-eps-hash OBJECT IDENTIFIER ::= {iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD4}

HashValue ::= SEQUENCE {
    hashAlgorithm DigestAlgorithmIdentifier,
    hashValue OCTET STRING
}

-- Create an S/MIME capability for advertising that
-- a client can understand the PLASMA recipient info
-- structure information
--
cap-Plasma-RecipientInfo SMIME-CAPS ::= {
    IDENTIFIED BY id-cap-plasma-recipientInfo
}

id-cap-plasma-recipientInfo OBJECT IDENTIFIER ::= {
    id-cap TBD5
}

END
Author’s Address

Jim Schaad
Soaring Hawk Consulting

Email: ietf@augustcellars.com