Network Working Group   J. Schaad
Internet-Draft   Soaring Hawk Consulting
Intended status: Standards Track   February 12, 2014
Expires: August 16, 2014

Plasma Service Cryptographic Message Syntax (CMS) Processing
draft-schaad-plasma-cms-05

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. A new model for enforcement of policy using a third party is described in RFC TBD [I-D.freeman-plasma-requirements]. This is the Policy Augmented S/MIME (PLASMA) system. This document provides the details needed to implement the new Plasma model in the CMS infrastructure.

An additional benefit of using the Plasma module is that the server, based on policy, manages who has access to the message and how the keys are protected.

The document details how the client encryption and decryption processes are performed, defines how to construct the CMS recipient info structure, a new content to hold the data required for the Plasma server to store the keys and policy information. The document does not cover the protocol between the client and the Plasma policy enforcement server. One example of the client/server protocol can be found in RFC TBD [I-D.schaad-plasma-service].

Status of This Memo

This Internet-Draft will expire on August 16, 2014.

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Vocabulary 4
  1.2. Requirements Terminology 4
2. Model 5
3. Recipient Info Encoding 5
  3.1. PLASMA Encrypted Key 7
  3.2. PLASMA Content Type 9
  3.3. CMS Signed Data signed attributes 14
    3.3.1. PLASMA URL Authenticated Attribute 14
    3.3.2. PLASMA Encrypted Content Hash Attribute 16
  3.4. Plasma Lockbox Attributes 17
    3.4.1. Audit Trail Identifier Attribute 17
    3.4.2. Signer Info Attribute 18
    3.4.3. XACML Generic Attribute 19
4. Sender Processing Rules 20
  4.1. Flow 20
5. Recipient Processing Rules 21
  5.1. Flow 21
  5.2. Reply Processing 23
6. S/MIME Capability 23
7. Mandatory Algorithms 24
  7.1. Plasma Servers 24
  7.2. Plasma Clients 24
8. Security Considerations 24
9. IANA Considerations 25
10. References 26
  10.1. Normative References 26
  10.2. Informative References 27
Appendix A. 2009 ASN.1 Module 28

Appendix B. Policy Encoding Techniques 32
Author's Address 33

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 the message. While this system works in theory, in practice it has some difficulties that have led 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 a Policy Augmented S/MIME (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 Plasma server is now responsible for obtaining valid keys for recipients and checking the policy access for the recipients. 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 are 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.

While this document describes the processes in terms of dealing with email system, a Plasma server can be used with any number of clients that need to protected content. Thus the same system could be used for protection of documents without having to specify in advance the individuals who should be able to open the document; it would just be allowed by the server based on the policy applied to the document.

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 [RFC5751].

o In Section 5 is a description of the modification from the recipient processing rules outlined in [RFC5751].

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): A key, usually a symmetric key, which is used to encrypt another key, usually a content encryption key.

1.2. Requirements Terminology

When capitalized, 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

Details on the model and the requirements for the Plasma system can be found in [I-D.freeman-plasma-requirements].

3. Recipient Info Encoding

In order for the Plasma system to function in CMS, a method needs to be chosen and described for how the CEK is to be protected and carried with the message, such that the recipient will be able to identified that this is a Plasma enabled message, know which Plasma server to contact and be able to get back the CEK needed to decrypt the message. Not all recipients of a message that has been encrypted using a Plasma server will need to contact the server in order to decrypt the message. There is nothing in what we are doing that prevents a message sender from building recipient info structures in a normal manner, except the possibly that the policy applied to the encrypted content could restrict it from happening. Additionally the Plasma server could return the standard recipient info structures to be added to the message for recipients, if it can pre-authorize them for access to the message and it can obtain the appropriate keying material.

There are two distinct methods that were considered for identifying a recipient info structure as being a Plasma enabled object. The first would be to define a new recipient info structure placed in the Other Recipient Info structure. The second option is to force the new recipient info structure into one of the existing strucutres.

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 recipient info 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.

Given the current state of implementations, it was determined that the second method would be used as it will work with more implementations. After implementation it might be found that using the first method is the better way to go, in that case the decision can be re-visited.

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, the Plasma recipient info can be thought of as a strange way to do the transformation and thus it kind of fits into the model. It is in a structure that will be supported by the most basic CMS implementiation and it is easy for client implementations to make the determination of a Plasma recipient info by looking at the OID in the key encryption algorithm identifier.

A recipient info structure as defined in this document MUST be created by a Plasma server and MUST NOT be created by client software. A protocol such as the one in RFC TBD1 [I-D.schaad-plasma-service] is used to transport the recipient info structure between the client and the server.

For the convenience of the reader we include the KEKRecipientInfo structure pieces here (copied from [RFC5911]):

```
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})}
```

For a Plasma KEKRecipientInfo structure, the fields are filled in as follows:

version is set to the value of 4.

kekid is a sequence where the fields are:

keyIdentifier contains the constant string "Plasma".

date is not used and is omitted.

other is not used and is omitted.

keyEncryptionAlgorithm contains the value id-kek-plasma-token. The parameter field MUST be omitted.

encryptedKey contains the Plasma Encrypted Key object. The details of this are covered in Section 3.1

3.1. PLASMA Encrypted Key

We defined a new Key Wrapping algorithm which uses the Plasma server to wrap the CEK in an encrypted lock box. In addition to the KEK, the lock box also contains the information that is needed by the Plasma 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 key wrap algorithm object for Plasma
--

kwa-plasma-lockbox KEY-WRAP ::= {
   IDENTIFIER id-alg-plasma-lockbox
   PARAMS ARE absent
   SMIME-CAPS { IDENTIFIED BY id-alg-plasma-lockbox }
}

-- SignedData IDENTIFIED BY id-keyatt-plasma-token

id-alg-plasma-lockbox OBJECT IDENTIFIER ::= {iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) alg(3) TBD2 }
```

We define a new KEW-WRAP object called kwa-plasma-lockbox. This key wrap algorithm is identified by the id-kwa-plasma-lockbox OID. The key wrap algorithm has no parameters and the parameters field MUST be absent the algorithm identifier is encoded. The encypted key object which is emitted by the algorithm is a 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-ct-authEnvelopedData.

eContent is CMS AuthEnvelopedData [RFC5083] 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.

authEncryptedContentInfo is a structure containing the following elements:

contentType is id-ct-plasma-LockBox.

contentEncryptionAlgorithm 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 PLASMA 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, the PLASMA hash attribute and the PLASMA url attributes.

* SHOULD contain the multiple signature attribute [RFC5752] if more than one signature exists.

* MAY contain the ESS security label attribute.

* other attributes can also be included.

When creating the recipient info structures for the AuthEnvelopedData structure, there will normally only need to be a single entry in the sequence as the only entity that needs to decrypt the PLASMA Lockbox is the Plasma 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 Plasma Service can be used.

There are a number of circumstances that a Plasma server would apply multiple signatures to a single lockbox. These circumstances include during key rollover while a certificate is approaching expiration, esp. if there is going to be a change in the trust anchor being used. Another circumstance would be if a new signature algorithm is being rolled out, having the old and the new algorithm on the message during the rollout period increases the chances that the signature can be validated. In these circumstances, the multiple signature attribute defined in RFC 5752 [RFC5752] allows for a client to determine that a signature has been removed which might be attempted as part of an attack to use a more insecure algorithm.

3.2. PLASMA Content Type

The inner-most content type for a Plasma Key Wrap Algorithm is a Plasma 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 content type is designed so that the Plasma server does not need to keep any state dealing with a message on the server itself. This allows for minimal information to be kept on the server, it only needs the state of it's current transactions, and the message can be processed by any of a number of servers without needing to replicate state about the message between them.

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

```
--
--  PLASMA Content Type
--

ct-plasma-LockBox CONTENT-TYPE ::= {
    TYPE PLASMA-LockBox
    IDENTIFIED BY id-ct-plasma-LockBox
}

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

PLASMA-LockBox ::= SEQUENCE {
```

```
         policy        OCTET STRING,
         keyList       KeyList,
         attrList      AttributeList OPTIONAL
      }

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

      NamedRecipient ::= SEQUENCE {
         recipientName       UTF8String, -- name of the recipient
         keyPolicy           [0] OCTET STRING OPTIONAL,
         keyIdentifier       OCTET STRING OPTIONAL,
         keyValue            [1] ProtectedKey,
         ...
      }

      ProtectedKey ::= CHOICE {
         cms                 [0] RecipientInfo,
         xml                 [1] OCTET STRING,
         ...
      }

      OneCek ::= SEQUENCE {
         keyPolicy           [0] OCTET STRING OPTIONAL,
         keyIdentifier       [1] OCTET STRING OPTIONAL,
         keyValue            OCTET STRING,
         ...
      }

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

      PlasmaLockboxAttributes ATTRIBUTE ::= {
           aa-plasma-AuditTrailIdentifier | aa-plasma-SignerInfo |
           aa-plasma-Xacml-Attribute, ... }
```

```
PlasmaSignedAttributes ATTRIBUTE ::= {
     aa-plasma-url | aa-plasma-econtent-hash
}
```

In the above ASN.1, the following items are defined:

ct-plasma-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 [RFC5911]). The content type associates the object identifier id-ct-plasma-LockBox with the data type PLASMA-LockBox.

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

PLASMA-LockBox is the new type defined for new content type. This is a sequence with the following fields:

policy contains the policy label that is to be applied to the KEK values in the keyList field. The exact content of the field will be specific to the set of Plasma servers involved. Servers MUST be able to deal with an XML encoding of the policy in this location. See Appendix B for some alternate encodings.

keyList contains the key values.

attrList contains a set of attributes which are considered as significant by the Plasma server internally. One example of an attribute that goes into this location is the audit trail identifier attribute. This attribute allows for an identifier to tagged to the message that can be used by all entities that are going to create entries in an audit log. Since they all have access to a unique identifier for this message, they can all use that identifier when creating their respective log entries for creation, granting of access and refusing access. The identifier can then be used to correlate all of these audit trail events back to a single message. This document defines three attributes to be placed in this location: Audit Trail Identifier Section 3.4.1, Signer Info Section 3.4.2 and XACML attribute Section 3.4.3.

KeyList is a new type that contains CEK values or 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 CEK 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.

defaultRecipients contains the CEK 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.

recipientName contains the name of the name of the recipient in the form of an RFC5321 email address.

keyPolicy contains a policy string specific to this key. If present the policy MUST be evaluated as accept before this recipient info structure is released. Servers MUST be able to deal with an XML encoding of the policy in this location. See Appendix B for some alternate encodings.

keyIdentifier contains the identification value for the CEK.

keyValue contains the encrypted key for the named recipient. The ProtectedKey structure is marked as extensible. If an unrecognized choice is made in the ProtectedKey structure, the NamedRecipient structure is to fail returning the key as it's type will not be recognized. There could be another named key with a different return type which can be returned successfully.

This structure is tagged as extensible; this was done because there may be a need to add additional fields such as other name types in the future.

ProtectedKey contains the CEK encrypted in some manner. The choice has the following fields:

cms contains a CMS recipient info structure for the named recipient.

xml contains an XML EncryptedKey structure from the XML Encryption standard [W3C.WD-xmlenc-core1-20101130].

The structure is marked as extensible. Servers MUST be able to deal with unrecognized encrypted key fields from future versions.

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

keyPolicy contains a policy string specific to this key. If present the policy MUST be evaluated as accept before this key is released. Servers MUST be able to deal with an XML encoding of the policy in this location. See Appendix B for some alternate encodings.

keyIdentifier contains the identification value for the CEK.

keyValue contains the CEK value.

This structure is tagged as extensible; this was done because there may be a need to add additional fields such as other name types in the future.

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

PLASMAAttributes defines an Object Set of attributes which can be included. The object set is intentionally open ended for later expansion. The object set is initialized with the three attributes defined in this document.

PlasmaSignedAttributes defines an Object Set of attributes that are intended for use as signed attributes for CMS SignedData objects. This item is intended to be added to the SignedAttributesSet in the CMS module in [RFC5911].

The recipientName field of the NamedRecipient structure is designed so that a client can build a CMS recipient info structure targeted to a specific recipient. In order for the Plasma server to know which of these named recipient structure to return it requires that the sender identify the recipient for the CMS recipient info structure and that the recipient identify itself so that the Plasma server can find the correct structure. We are using Email names in the form of internationalized RFC 5321 [RFC5321] address names. There are a number of issues that are associated with the use of this name form for comparison purposes. As stated in Section 2.3.11 of RFC 5321,

> the local-part MUST be interpreted and assigned semantics only by the host specified in the domain part of the address.

While many platforms do case-insensitive comparisons of mailbox names, there is not a way for an independent server to know if this is appropriate behavior. A similar issue exists with Unicode normalization as pointed out in Section 10.1 of RFC 6530 [RFC6530]. The server that holds the mailbox can have a consistent rule for normalization, but a Plasma server in separate domain may not know the appropriate rules to use.

Plasma servers SHOULD do the following when comparing the Email addresses found in the recipientName field:

1. The domain name portion is compared using procedure in Section 2.3.2.4 of [RFC5890]. The rules are:

   * Exact (bit-string identity) matches between pairs of U-labels.

   * Matches between a pair of A-labels, using normal DNS case-insensitive matching rules.

   * Equivalence between a U-label and an A-label determined by translating the U-label form into an A-label form and then testing for a match between the A-labels using normal DNS case-insensitive rules.

2. The local name portion of the name is compared using bit-string identity. Plasma servers MAY apply appropriate transformations for local domain names, Plasma servers MAY provide for administration configuration to allow for appropriate transformations to non-local domains, but SHOULD NOT apply any transformations in the absence of administrative configureation.

3.3. CMS Signed Data signed attributes

3.3.1. PLASMA URL Authenticated Attribute

It is required that the name of the Plasma 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-plasma-url ATTRIBUTE ::= {
   TYPE UTF8String IDENTIFIED BY id-aa-plasma-url
}

id-aa-plasma-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-plasma-url has been defined.

The OID value of id-aa-plasma-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 Plasma Server. The URL defines both the destination server and the protocol to be used. When the schema for the URL is "plasma", then the protocol used is [I-D.schaad-plasma-service].

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 [RFC5911].

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 unique. Multiple values are allowed as there can be multiple Plasma servers that can be used to evaluate the policy. Since the URLs will be sorted during encoding, 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 a Plasma Server. There are no processing rules for the sender of a message. The processing rules for a recipient can be found in Section 5.

3.3.2. PLASMA 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-plasma-econtent-hash ATTRIBUTE ::= {
   TYPE HashValue IDENTIFIED BY id-aa-plasma-econtent-hash
}

id-aa-plasma-econtent-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-plasma-econtent-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 mdoule contained in [RFC5911].

id-aa-plasma-econtent-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.

3.4. Plasma Lockbox Attributes

3.4.1. Audit Trail Identifier Attribute

The Audit Trail Identifier attribute allows for a unique and persistent identifier to be carried as part of a Plasma Lockbox message. This identifier can then be used by Plasma servers when creating log entries in the audit trail to designate a single Plasma message. This use of a single identifier allows for better correlation to occur by auditors, however as the identifier is hidden from all viewers except the Plasma server, the message itself is not locatable from the log entries.

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

```
--
--  Attribute to hold an Audit Trail Identifier
--

aa-plasma-AuditTrailIdentifier ATTRIBUTE ::= {
  TYPE OCTET STRING
  IDENTIFIED BY id-aa-plasma-Audit-Trail-Identifier
}

id-aa-plasma-Audit-Trail-Identifier OBJECT IDENTIFIER ::= {
     iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD6}
```

In this ASN.1, the following items are defined:

aa-plasma-AuditTrailIdentifier
   This is an object of type ATTRIBUTE that associates the identifier id-aa-plasma-Audit-Trail-Identifier with the type OCTET STRING. When used in attrList field of a PLASMA-LockBox, the values set MUST contain a single value. The value is the audit trail identifier value.

id-aa-plasma-Audit-Trail-Identifier
   This is the OID used to identifier this attribute.

The use of OCTET STRING for the content allows for the greatest flexibility for Plasma Servers in devising the value to use. The content of the Audit Tail Identifier is intended to be an opaque value to all entities, all Plasma servers MUST be able to ignore how the value is structured.

3.4.2. Signer Info Attribute

Some policies require that the inner content of an encrypted message be signed as well. However the encrypted data stream of the message is not provided to the Plasma server for it to verify that it was done successfully. The only places to check is in the audit trail for the message and/or to allow the client to do the check that the signature is present. This attribute provides a location for the Plasma server to place the provided CMS SignerInfo structure(s) provided by the client to be carried with the message. The server can then push the structure(s) to the client and the client can validate that the actual signatures on the message match the signatures provided by the server. All servers MUST be able to parse this attribute and convert it to an appropriate XACML attribute to return to clients.

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

```
--
--  Attribute to hold a SignerInfo structure
--

aa-plasma-SignerInfo ATTRIBUTE ::= {
  TYPE SignerInfo IDENTIFIED BY id-aa-plasma-signerInfo
}

id-aa-plasma-signerInfo OBJECT IDENTIFIER ::= {iso(1)
     member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD7}
```

In this ASN.1, the following items are defined:

aa-plasma-SignerInfo
   This is an object of type ATTRIBUTE that associates the identifier id-aa-plasma-SignerInfo with the type SignerInfo.

id-aa-plasma-SignerInfo
   This OID is used to identify the attribute, it's associated type and it's semantics.

There can be one or more attribute values in the attribute set. Each of the values is to be treated independently and returned to the client. The values may be returned in a single Attributes XML element.

3.4.3. XACML Generic Attribute

Many times Plasma servers be in situation where they will need to return various values to the clients. These decisions will frequently be taken by the originating Plasma server, since it may be the only one that has the data to be returned. This attribute allows for any data to be carried in the form of an XACML [XACML] attribute XML structure. Since the content is an XACML attribute, it can be pushed to the client without the client needing to understand or evaluate the content being presented. The Signer Info attribute presented in the previous section could have been implemented using this attribute rather than defining it's own attribute, however the space savings was deemed sufficient to justify the creation of the new attribute.

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

```
--
--  Attribute to hold an arbitrary XACML XML attribute
--  structure
--

aa-plasma-Xacml-Attribute ATTRIBUTE ::= {
  TYPE OCTET STRING IDENTIFIED BY id-aa-plasma-Xacml-Attribute
}

id-aa-plasma-Xacml-Attribute OBJECT IDENTIFIER ::= {iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD8}
```

In this ASN.1, the following items are defined:

aa-plasma-Xacml-Attribute
   This is an object of type ATTRIBUTE that associates the identifier of id-aa-plasma-Xacml-Attribute with the type OCTET STRING.

id-aa-plasma-Xacml-Attribute
   This OID is used to identify the attribute, its associated type and it's semantics.

There can be one or more attributes values associated with the attribute set. Each of the values is to be treated independently and returned as separate items to the client.

The data type is an OCTET STRING to allow for alternate XML encodings to be used. All servers MUST be able to deal with a UTF8 string XML encoding of the policy in this location. See Appendix B for alternate encoding methods. If a server cannot understand the encoding presented, the server MUST fail processing of the lockbox. If the server cannot understand the attribute, and the attribute is required for processing the access control statement, the server MUST fail that portion of the access control evaluation.

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 selects the content encryption algorithm (with input from the policies chosen) and randomly creates the CEK.

6. Sender Agent encrypts the content with the CEK and computes the encrypted hash value.

7. Sender Agent may optionally create lock boxes for one or more message recipients. (These are for the early-bind recipients that are protected by the policy server.)

8. Sender Agent transmits the CEK, the list of recipients, the set of policy protected recipient lock boxes, the encrypted hash value and the policy label to the PLASMA server.

9. Sender Agent receives a set of recipient info structures from the policy server. If the policy validation fails then the sender agent cannot send the message under the current policy label.

10. Sender Agent verifies the signature on the signed data structure inside of the PLASMA-KEK attribute.

    A. Signature is current and passes cryptographic processing.

    B. Signed attributes contains the PLASMA URL attribute and the PLASMA Encrypted Hash attribute.

    C. The certificate used to validate the signature MUST have the Plasma XXXX EKU (defined in Section X.Y of RFC XXXX).

    D. Other standard signature checks.

    The Sender Agent SHOULD validate all of the signatures if more than one signature exists.

11. Sender Agent adds the recipient info structures returned from the Plasma server to those it creates for early bind recipients which are not protected by policy.

12. Sender Agent finishes encoding the message and transmits it to the MTA.

5. Recipient Processing Rules

5.1. Flow

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 Paragraph 5 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 a PLASMA-KEK attribute. It is recommended that the entire list of recipient info structures be checked for one that can be processed directly before processing a Plasma receipient structure.

3. The Receiving Agent validates the PLASMA-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 needs to succeed only on one SignerInfo element, 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 PLASMA URL attribute.

B. The certificate used to validate the signature MUST contain the XXXX value in the EKU extension. The certificate MUST NOT contain the anyPolicy value in the EKU extension. Local policy can dictate that content of the Plasma URL attribute be used in selecting trust anchors for the signing certificate.

C. If an ESS security label attribute ([RFC2634]) is present, then it must be evaluated and processing ends if the security label processing fails or is denied.

D. If the PLASMA URL attribute is absent, then processing fails.

E. The URL value in the PLASMA 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 a random Plasma server. The schema of the URL MUST be one that the client implements. (For example the "plasma" schema associated with RFC XXX [I-D.schaad-plasma-service].) As discussed in Section 4.5 of [I-D.freeman-plasma-requirements], policy can be enforced on the edge of an enterprise, this means that if multiple URLs are present in the Plasma URL attribute they all need to be checked for policy and ability to use before this step fails.

F. The PLASMA Encrypted Hash attribute value is checked against the encrypted content. If this attribute is absent then processing fails. If the value does not matched the computed value on the encrypted content 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 Plasma server and passes in the identity and attribute statements and receives back the CEK or a lock box to allow it to obtain the CEK value.

5.2. Reply Processing

In some circumstances a message recipient may be permitted to read a message sent under a certan 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 [RFC5751] 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
--     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
}
```

We define a new SMIME-CAPS object called cap-Plasma-RecipientInfo. This attribute is identified by the 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

7.1. Plasma Servers

Servers MUST implement AES-GCC-128 [RFC5084] for the content encryption algorithm in section 3.1. Other authenticated encryption algorithms MAY be implemented.

Servers MUST implement RSA v1.5 as a key transport algorithm for lockboxes created in section 3.1 and for pre-authenticated lock boxes returned in step 8 of section 4.1. Servers SHOULD implement RSA OAEP as a key transport algorithm in the same locations. Other key transport algorithms MAY be implemented.

Servers MUST implement EC-DH as a key agreement algorithm for lockboxes created in section 3.1 and for pre-authenticated lock boxes returned in step 8 of section 4.1. Servers MAY implement other key agreement algorithms.

Servers MUST implement the RSA v1.5 signature algorithm with SHA-256 and SHA-512. Servers MUST implement the EC-DSA signature algorithm with SHA-256 and SHA-512 for producing signature on the Plasma lock box. Other signature algorithms MAY be implemented as well.

7.2. Plasma Clients

Clients MUST implement the mandatory algorithms defined for S/MIME [RFC5751] for the lock boxes created in step 7 and transmitted to the server in step 8 of Section 4. Other algorithms MAY be implemented.

Clients MUST implement SHA-256 and SHA-512 for computation of the Plasma Encrypted Content Hash in section 3.4. Other algorithms MAY be implemented, but doing so can cause clients that do not implement this algorithm to not attempt to read the message.

When verifying signatures on the Plasma lock boxes, clients MUST be able to verify the RSA v1.5 signature algorithm with SHA-256 and SHA-512. Clients MUST also be able to verify the EC-DSA signature algorithm with SHA-256 and SHA-512 signature algorithm. Clients MAY be able to verify other signature algorithms.

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.

Still more expansion....

The hash computed for the Plasma Encrypted Content Hash attribute has different security concerns that a hash used for signature computation. This hash value is used to get a degree of assurance that the encrypted content is associated with Plasma lock box. In the event that a collision exists, then the client will go and talk to the server to get a content encryption key when that key will not successfully decrypt the content. However this does not affect the privacy of the client as the client's decision to talk to the server is based on the URL(s) of the server and the validation of the server's signature. This means that an attacker that substitutes an encrypted content needs not only to have the hash of the encrypted content be correct, but the decrypted content needs to make sense. In order for an attacker to have the client talk to it, it needs to attack the certificates or signature produced on the lock box and not the encrypted content itself.

9. IANA Considerations

This document requires that a number of Object Identifiers be assigned. These are now under the control of IANA following [I-D.housley-smime-oids].

IANA is requested to assign the following identifiers:

- TBD9 is to be assigned from the "SMI Security for S/MIME Module Identifer" registry. The description for the entry is suggested as id-mod-plasma-2013.

- TBD1 is to be assigned from the "SMI Security for S/MIME CMS Content Type" registry. The description for the entry is suggested as id-ct-plasma-LockBox.

- TBD2 is to be assigned from the "SMI Security for S/MIME Algorithms" registry. The description for the entry is suggested as id-alg-plasma-lockbox.

- TBD3 is to be assigned from the "SMI Security for S/MIME Attributes" registry. The description for the entry is suggested as id-aa-plasma-url.

- TBD4 is to be assigned from the "SMI Security for S/MIME Attributes" registry. The description for the entry is suggested as id-aa-plasma-econtent-hash.

o TBD5 is to be assigned from the "SMI Security for S/MIME Capabilities" registry. The description for the entry is suggested as id-cap-plasma-recipientInfo.

o TBD6 is to be assigned from the "SMI Security for S/MIME Attributes" registry. The description for the entyr is suggested as id-aa-plasma-Audit-Trail-Identifier.

o TBD7 is to be assigned from the "SMI Security for S/MIME Attributes" registry. The description for the entry is suggested as id-aa-plasma-signerInfo.

o TBD8 is to be assigned from the "SMI Security for S/MIME Attributes" registry. THe description for the entry is suggested as id-aa-plasma-Xacml-Attribute.

10. References

10.1. Normative References

[RFC5911] Hoffman, P. and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, June 2010.

[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.

[RFC2634] Hoffman, P., "Enhanced Security Services for S/MIME", RFC 2634, June 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC5752] Turner, S. and J. Schaad, "Multiple Signatures in Cryptographic Message Syntax (CMS)", RFC 5752, January 2010.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC6530] Klensin, J. and Y. Ko, "Overview and Framework for Internationalized Email", RFC 6530, February 2012.

[I-D.freeman-plasma-requirements]
Freeman, T., Schaad, J., and P. Patterson, "Requirements for Message Access Control", draft-freeman-plasma-requirements-08 (work in progress), October 2013.

[W3C.WD-xmlenc-core1-20101130]
Roessler, T., Reagle, J., Hirsch, F., and D. Eastlake, "XML Encryption Syntax and Processing Version 1.1", World Wide Web Consortium LastCall WD-xmlenc-core1-20101130, November 2010, <http://www.w3.org/TR/2010/WD-xmlenc-core1-20101130>.

10.2. Informative References

[RFC3369] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3369, August 2002.

[RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[RFC4262] Santesson, S., "X.509 Certificate Extension for Secure/Multipurpose Internet Mail Extensions (S/MIME) Capabilities", RFC 4262, December 2005.

[RFC5084] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, November 2007.

[I-D.schaad-plasma-service]
Schaad, J., "Plasma Service Trust Processing", draft-schaad-plasma-service-04 (work in progress), January 2013.

[XACML] Rissanen, E., Ed., "eXtensible Access Control Markup Language (XACML) Version 3.0", OASIS Standard xacml-201008, August 2010, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01.en.doc>.

[EXI] Kamiya, T. and J. Schneider, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium CR CR-exi-20091208, December 2009, <http://www.w3.org/TR/2009/CR-exi-20091208>.

[I-D.housley-smime-oids]
Housley, R., "Object Identifier Registry for the S/MIME Mail Security Working Group", draft-housley-smime-oids-00 (work in progress), October 2013.

Appendix A. 2009 ASN.1 Module

```
PolicySMime-2008 { iso(1) member-body(2) us(840) rsadsi(113549)
       pkcs(1) pkcs-9(9) smime(16) modules(0)
       id-mod-plasma-2013(TBD9) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
  IMPORTS
   -- Cryptographic Message Syntax (CMS) [RFC5652]

   CONTENT-TYPE, RecipientInfo, SignedData,
   DigestAlgorithmIdentifier, SignerInfo, KEY-WRAP
   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) }
   ;

   --
```

```
--  PLASMA Content Type
--

ct-plasma-LockBox CONTENT-TYPE ::= {
    TYPE PLASMA-LockBox
    IDENTIFIED BY id-ct-plasma-LockBox
}

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

PLASMA-LockBox ::= SEQUENCE {
   policy       OCTET STRING,
   keyList      KeyList,
   attrList     AttributeList OPTIONAL
}

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

NamedRecipient ::= SEQUENCE {
   recipientName       UTF8String, -- name of the recipient
   keyPolicy           [0] OCTET STRING OPTIONAL,
   keyIdentifier       OCTET STRING OPTIONAL,
   keyValue            [1] ProtectedKey,
   ...
}

ProtectedKey ::= CHOICE {
   cms                 [0] RecipientInfo,
   xml                 [1] OCTET STRING,
   ...
}

OneCek ::= SEQUENCE {
```

```
       keyPolicy           [0] OCTET STRING OPTIONAL,
       keyIdentifier       [1] OCTET STRING OPTIONAL,
       keyValue            OCTET STRING,
       ...
    }

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

    PlasmaLockboxAttributes ATTRIBUTE ::= {
         aa-plasma-AuditTrailIdentifier | aa-plasma-SignerInfo |
         aa-plasma-Xacml-Attribute, ... }

    PlasmaSignedAttributes ATTRIBUTE ::= {
         aa-plasma-url | aa-plasma-econtent-hash
    }


    --
    --  New key wrap algorithm object for Plasma
    --

    kwa-plasma-lockbox KEY-WRAP ::= {
       IDENTIFIER id-alg-plasma-lockbox
       PARAMS ARE absent
       SMIME-CAPS { IDENTIFIED BY id-alg-plasma-lockbox }
    }

    -- SignedData IDENTIFIED BY id-keyatt-plasma-token

    id-alg-plasma-lockbox OBJECT IDENTIFIER ::= {iso(1) member-body(2)
         us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) alg(3) 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-plasma-url ATTRIBUTE ::= {
       TYPE UTF8String IDENTIFIED BY id-aa-plasma-url
    }

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

id-aa-plasma-econtent-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
}

--
--  Attribute to hold an Audit Trail Identifier
--

aa-plasma-AuditTrailIdentifier ATTRIBUTE ::= {
  TYPE OCTET STRING
  IDENTIFIED BY id-aa-plasma-Audit-Trail-Identifier
}

id-aa-plasma-Audit-Trail-Identifier OBJECT IDENTIFIER ::= {
     iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD6}
```

```
    --
    --  Attribute to hold a SignerInfo structure
    --

    aa-plasma-SignerInfo ATTRIBUTE ::= {
      TYPE SignerInfo IDENTIFIED BY id-aa-plasma-signerInfo
    }

    id-aa-plasma-signerInfo OBJECT IDENTIFIER ::= {iso(1)
         member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD7}


    --
    --  Attribute to hold an arbitrary XACML XML attribute
    --  structure
    --

    aa-plasma-Xacml-Attribute ATTRIBUTE ::= {
      TYPE OCTET STRING IDENTIFIED BY id-aa-plasma-Xacml-Attribute
    }

    id-aa-plasma-Xacml-Attribute OBJECT IDENTIFIER ::= {iso(1)
        member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) TBD8}


 END
```

Appendix B. Policy Encoding Techniques

This appendix is informative.

The fields for encoding a policy expression is an ASN.1 OCTET STRING. This field type was chosen so that servers would have the widest choice of methods to encode the policy expressions. For stand alone servers, the only issue is that the server will be able to correctly extract and use the policy expression, as such it can be kept in XML or converted into a format that is more natural to the policy evaluation engine used by the server. When one wants to use multiple servers, then all of the servers involved need to be able to use the encoded format(s) and re-map them into the internal versions that are used locally. This is far more complicated when the servers are hosted by different organizations that might be using different local policy evaluation engines.

It is RECOMMENDED that what ever encoding method is used normally, a provision exist for the XML version of the policy string as presented in RFC XXX [I-D.schaad-plasma-service] exist without change. Doing so allows for a single common format to be shared among all Plasma

servers independent of the organization providing the server and the one operating the server. The server will be able to determine the set of other servers that will be able to process the content, as the server must be configured with that information in order to create the appropriate lock boxes for the other servers to access the encrypted content.

There are two different methods that exist where the XML encoding can be compressed before placing it into the OCTET STRING. The first would be to use the Efficient XML Interchange (EXI) Format documented in [EXI]. A second method would be to use the standard DEFLATE algorithm either with or without a pre-defined library.

A possible method of encoding would to be use the first byte to identify the encoding technique, reserving 0x3C for vanilla XML strings. Following bytes could be used to determine which pre-defined table was used and then the compressed encoding.

Author's Address

Jim Schaad
Soaring Hawk Consulting

Email: ietf@augustcellars.com