Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2014

J. Schaad
Soaring Hawk Consulting

February 14, 2014

Plasma Service Trust Processing
draft-schaad-plasma-service-05

Abstract

RFC TBD describes a new model and set of requirements to implement a labeling system on Cryptographic Message Syntax (CMS) objects where the entity in charge of doing the label enforcement is under the control of a central authority rather than the recipient of the object.

This document describes a protocol to be used by senders and recipients of CMS objects to communicate with a centralized label enforcement server. The document outlines how a client will get the set of labels or policies that it can use for sending messages, composes a secure CMS object with a label on it and gets the necessary keys to decrypt a CMS object from the server. This document is designed to be used with RFC TBD2 which describes the extensions used in CMS objects to hold the label information.

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 August 18, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

RFC TBD [I-D.freeman-plasma-requirements] describes a new model and set of requirements to implement a labeling system on Cryptographic Message Syntax (CMS) objects where the entity in charge of doing the label enforcement is under the control of a central authority rather than the recipient of the object.

This document describes a protocol to be used by senders and recipients of CMS objects to communicate with a centralized label enforcement server. The document outlines how a client will get the set of labels or policies that it can use for sending messages, composes a secure CMS object with a label on it and gets the

necessary keys to decrypt a CMS object from the server. This document is designed to be used with RFC TBD [I-D.schaad-plasma-cms] which describes the extensions used in CMS objects to hold the label information.

1.1. XML Nomenclature and Name Spaces

The following name spaces are used in this document:

Prefix	Namespace	Specification(s)
eps	http://ietf.org/2011/plasma/	This Specification
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512	[WS-TRUST]
xac ml	http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.html	[XACML]
ds2	http://www.w3.org/2000/09/xmldsig#	[XML-Signature]
xs	http://www.w3.org/2001/XMLSchema	[XML-Schema1][XML-Schema2]

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].

When the words appear in lower case, their natural language meaning is used.

2. Components

In designing this specification we used a number of pre-existing specifications as building blocks. In some cases we use the entirety of the specification and in other case we use only select pieces.

2.1. XACML 3.0

The XACML specification (eXtensible Access Control Markup Language) [XACML] provides a framework for writing access control policies and for creating standardized access control queries and responses. The request and response portion of the specification is used to build the request (Section 5.2) and response (Section 6.1) messages in this specification. The structure for writing the access control policies is out of scope for this document, but XACML is one of the possibilities that can be used for that purpose.

2.2. SAML

A number of different methods for carrying both identification and attributes of the party requesting access is permitted in this specification. SAML is one of the methods that is permitted for that purpose.

SAML has defined three different types of assertions in it's core specification [OASIS-CORE]:

- Authentication: The assertion subject was authenticated by a particular means at a particular time.

- Attribute: The assertion subject is associated with the supplied attributes.

- Authorization Decision:[[CREF1: I don't see Plasma using this type --Trevor]] A request to allow the assertion subject to access the specified resource has been granted or denied.

While a PDP can use an Authorization Decision as input, this is unexpected and MAY be supported. In addition there are three different ways that the subject of a SAML statement can be identified:

- A bearer statement: These statements are belong to anybody who presents them. The owner is required to take the necessary precautions to protect them.

- A holder of key statement: These statements belong to anybody who can use the key associated with the statement.

- Subject match:[[CREF2: What about attribute match? --Trevor]] These statements can be associated to an identity by matching the name of the entity.

We cannot pass a SAML assertion with attributes as a single attribute in the XACML request as XACML wants each of the different attributes to be individually listed in the request. This greatly simplifies the XACML code, but means that one needs to do a mapping process from the SAML attributes to the XACML attributes. This process has been discussed in Section 2 of [SAML-XACML]. This mapping process MUST be done by a trusted agent, as there are a number of steps that need to be done including the validation of the signature on the SAML assertion. This process cannot be done by the PEP that is residing on the Plasma client's system as this is considered to be an untrusted entity by the Plasma system as a whole. One method for this to be addressed is to treat the Plasma server as both a PDP (for the Plasma client) and a PDP for the true XACML policy evaluator. In this model, the Plasma server becomes the trusted PEP party and has the ability to do the necessary signature validation and mapping processes. A new XACML request is then created and either re-submitted to itself for complete evaluation or to a third party which does the actual XACML processing.[[CREF3: This sounds like we ignore the mapping on the wire. There is no reason to mandate the mapping occurs inside the PDP. --Trevor]]

2.3. WS-Trust 1.4

The WS-Trust 1.4 [WS-TRUST] standard provides for methods for issuing, renewing, and validating security tokens. This specification uses only a small portion of that standard, specifically the structure that returns a trust token from the issuer to the requester.

This specification makes no statements on the content and format of the token returned from the Plasma server to the Plasma client in the wst:RequestSecurityTokenResponse field. These tokens may be parseable by the client, but there is no requirement that the client be able to understand the token. The token can always be treated as an opaque blob by the client which is simply reflected back to the server at a later time. The attributes that client needs to understand in order to use the token, such as the life time, are returned as fields of the token response.

TODO: need to discuss the content model and say what elements need to be supported and what elements can be ignored -- safely.

3. Model

To be supplied from the problem statement document. [[CREF4: Should one be able to create a policy on the fly for specific item where a set of attributes can be defined by the sender of the message. --Brian]]

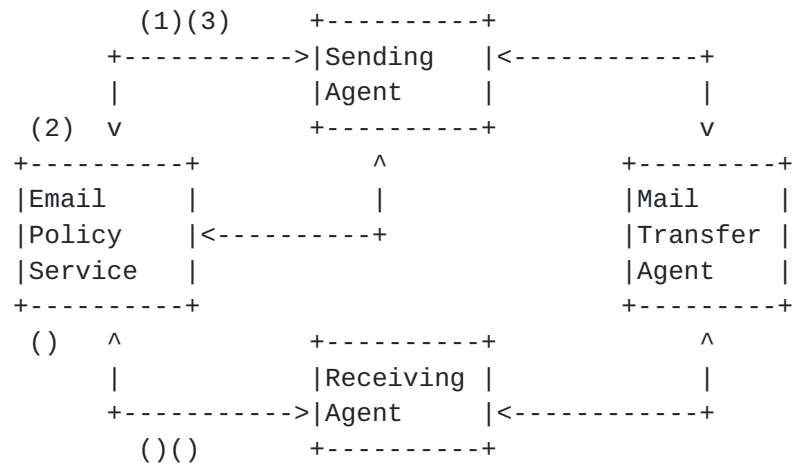

Figure 1: Message Access Control Actors

List the boxes above and give some info about them.

Email Policy Service is the gateway controller for accessing a message. Although it is represented as a single box in the diagram, there is no reason for it to be in practice. Each of the three protocols could be talking to different instances of a common system. This would allow for a server to operated by Company A, but be placed in Company B's network thus reducing the traffic sent between the two networks.

Mail Transfer Agent is the entity or set of entities that is used to move the message from the sender to the receiver. Although this document describes the process in terms of mail, any method can be used to transfer the message.

Receiving Agent is the entity that consumes the message.

Sending Agent is the entity that originates the message.

3.1. Sender Processing

We layout the general steps that need to be taken by the sender of an EPS message. The numbers in the steps below refer to the numbers in the upper half of Figure 1. A more detailed description of the processing is found in Section 7 for obtaining the security policies that can be applied to a messages and Section 8 for sending a message.

1. The Sending Agent sends a message to one or more Email Policy Services in order to obtain the set of policies that it can apply to a message along with a security token to be used in proving

the authorization. Details of the message send can be found in Section 7.1.

2. The Email Policy Service examines the set of policies that it understands and checks to see if the requester is authorized to send messages with the policy.

3. The Email Policy Service returns the set of policies and an security token to the Sending Agent. Details of the message sent can be found in Section 7.2.

4. The Sending Agent selects the Email Policy(s) to be applied to the message, along with the set of recipients for the message.

5. The Sending Agent relays the selected information to the Email Policy Service along with the security token. Details of this message can be found in Section 8.1.

6. The Email Policy Service creates the recipient info attribute as defined in [I-D.schaad-plasma-cms].

7. The Email Policy Service returns the created attribute to the Sending Agent. Details of this message can be found in Section 8.2.

8. The Sending Agent composes the CMS EnvelopedData content type placing the returned attribute into a KEKRecipientInfo structure and then send the message to the Mail Transport Agent.

3.2. Recieving Agent Processing

We layout the general steps that need to be taken by the sender of an EPS message. The numbers in the steps below refer to the numbers in the lower half of Figure 1. A more detailed description of the processing is found in Section 9.

1. The Receiving Agent obtains the message from the Mail Transport Agent.

2. The Receiving Agent starts to decode the message and in that process locates an EvelopedData content type which has a KEKRecipientInfo structure with a XXXX attribute.

3. The Receiving Agent processes the SignedData content of the XXXX attribute to determine that communicating with it falls within accepted policy.

4. The Receiving Agent transmits the content of the XXXX attribute to the referenced Email Policy Service. The details of this message can be found in Section 9.1.

5. The Email Policy Service decrypts the content of the message and applies the policy to the credentials provided by the Receiving Agent.

6. If the policy passes, the Email Policy Service returns the appropriate key or RecipientInfo structure to the Receiving Agent. Details of this message can be found in Section 9.2.

7. The Receiving Agent proceeds to decrypt the message and perform normal processing.

4. Protocol Overview

The protocol defined in this document is designed to take place between a Plasma server and a Plasma client. The protocol takes place in terms of a request/response dialog from the client to the server. A single dialog can consist of more than one request/response message pair. Multiple round trips within allow a client to provide additional authentication, authorization and attribute information to the server.

Each dialog contains one or more action attributes specifying what actions the client wishes the server to take. Depending on the action requested, additional attributes may be present providing data for the action to use as input. Finally, each dialog will contain authentication and attributes supplied by one or more authorities that the server can use either as input to the action or as input to policy decisions about whether to perform the action.

The protocol MUST be run over a secure transport, the secure transport is responsible for providing the confidentiality and integrity protection services over the entire message. The protocol allows for signature operations to occur on sub-sections of the message structure, howewever this is used for creation of identity proofs and not for integrity protection.

Multiple dialogs may be run over a single secure transport session. Before a new dialog may be started, the previous dialog MUST have completed to a state of success, failure or not applicable. A new dialog MUST NOT be started after receiving a response with an indeterminate status. If a new dialog is desired in these circumstances, then the transport session MUST to be closed and re-opened. [[CREF5: --- I want to say that TLS reconnect using caching is OK here. Is that a reasonable statement? I don't think we want

to say that the server will keep Plasma session data across TLS sessions. --JLS]]

5. Plasma Request

The specification is written using XACML as the basic structure to frame a request for an operation. The request for operations to occur are written using XACML action items. This specification defines actions specific to Plasma in a CMS environment. Other specifications can define additional action items for other environments (for example the XML encryption environment) or other purposes. (Future work could use this basic structure to standardize the dialogs between PDPs and PAPs or to facilitate legal signatures on emails.)

In addition to the XACML action request there is a set of structures to allow for a variety of authentication mechanisms to be used. By allowing for the use of SAML and GSS-API as base authentication mechanisms, the mechanism used is contained in a sub-system and thus does not directly impact the protocol.

The request message uses a single XML structure. This structure is the eps:PlasmaRequest object. The XML Schema used to describe this structure is:

```
<xs:element name="PlasmaRequest" type="eps:RequestType"/>
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element ref="eps:Authentication" minOccurs="0"/>
    <xs:element ref="xacml:Request"/>
  </xs:sequence>
  <xs:attribute name="Version" type="xs:string" default="1.0"/>
</xs:complexType>
```

The RequestType has two elements in it:

Authentication is an optional element that holds the structures used for doing authentication and authorization. Unless no authentication is required by the Plasma server, the element is going to exist for one or more requests in the dialog.

xacml:Request is a required element that contains the control information for the action requested. The control information takes the form of an action request plus additional data to be used as part of the action request. The data and actions are to be treated as self-asserted, that is they are deemed not to come from a reliable source even in the event that an authentication is successfully completed. As self-asserted values, Plasma servers

need to exercise extreme care about which are included in the policy enforcement decisions. As an example, it makes sense to allow for the action identifier to be included in the policy enforcement, but assertions about the identity of the subject should be omitted. This element is taken from the XACML specification.

For some operations, display string values are returned as part of the response from the server. The xml:lang attribute SHOULD be included in the RequestType element to inform the server as to what language client wishes to have the strings in. The server SHOULD attempt to return strings in the language requested or a related language if at all possible.

5.1. Authentication Element

One of the major goals in the Plasma work is to detach the process of authentication specifics from the Plasma protocol. In order to accomplish this we are specifying the use of two general mechanisms (SAML and GSS-API) which can be configured and expanded without changing the core Plasma protocol itself. The authentication element has two main purposes: 1) to process the authentication being used by the client and 2) to carry authenticated attributes for use in the policy evaluation.

When transporting the authentication information, one needs to recognize that there may be a single or multiple messages in the dialog in order to complete the authentication process. In performing the process of authenticating, any or all of the elements in this structure can be used. If there are multiple elements filled out, the server can choose to process the elements in any order. This means that the Plasma protocol itself does not favor any specific mechanism. The current set of mechanisms that are built into the Plasma specification are:

- SAML Assertions - many different types of SAML assertions are supported. The specification uses both bearer and holder of key assertions.

- X.509 Certificates can be used for the purpose of authentication by creating a signature with the XML Digital Signature standard.

- GSS-API - the specification allows for the use of GSS-API in performing the authentication process. The ABFAB mechanism in GSS-API is specifically designed for use in a federated community and allows for both authentication and attribute information to be queried from the identity server.

- WS-Trust tokens allow for much of the same type of information to be passed as SAML assertions. The Plasma specification has been designed mainly for the use of WS-Trust tokens to be used for caching prior authentication sessions.

More than one authentication element can be present in any single message. This is because a client may need to provide more than one piece of data to a server in order to authenticate, for example a holder of key SAML Assertion along with a signature created with that key. Additionally a client may want to provide the server an option of different ways of doing the authentication. In a federated scenario, an X.509 certificate with a signature can be presented and the server may not be able to build a trust path to it's set of trust anchors. In this case the client may need to use the GSS-API/EAP protocol for doing the authentication. The client may want to provide the server with one or more SAML Assertion that binds a number of attributes to it's identities so that the server does not need to ask for those attributes at a later time. Finally, multiple entities may need to be validated (for example the user and the user's machine).

When transporting the attribute information, one needs to recognize that there may be single or multiple messages in the dialog in order to complete the authorization process. The server will return a status code of urn:oasis:names:xacml:1.0:status:missing-attribute in the event that one or more attributes are needed in order to complete the authorization process. The details on how XACML returns missing attribute information is found in Section 7.17.3 of [XACML]. When the list of attributes is returned, the client has two choices: 1) It can close the dialog, look for a source of the missing attributes and then start a new dialog, 2) it can just get an assertion for the missing attributes and send the new assertion as in a new request message within the same dialog. The decision of which process to use will depend in part on how long it is expected to take to get the new attribute assertion to be returned.

The same authentication data does not need to be re-transmitted to the server in a subsequent message within a single dialog. The server MUST retain all authenticated assertion information during a single dialog.

The schema for the Authentication element directly maps to the ability to hold the above elements. The schema for the Authentication element is:

```
<xs:element name="Authentication" type="eps:AuthenticationType"/>
<xs:complexType name="AuthenticationType">
  <xs:choice maxOccurs="unbounded">
    <xs:element ref="saml:Assertion"/>
    <xs:element name="GSSAPI" type="xs:hexBinary"/>
    <xs:element name="RoleToken">
      <xs:complexType>
        <xs:sequence>
          <xs:any namespace="##any" processContents="lax"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element ref="ds2:Signature"/>
    <xs:element name="Other">
      <xs:complexType>
        <xs:sequence>
          <xs:any namespace="##other"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

The schema allows for multiple authentication elements to occur in any order. It is suggested, but not required, that the ds2:Signature element occur after the authentication element that has an assoicated key. This makes it easier for servers to make a one pass validate of all authentication elements.

The Other element is provided to allow for additional authentication elements, include SAML version 1.1, to be used.

5.1.1. SAML Assertion

SAML Assertions can provide authentication or attribute information to the server. A SAML statement only needs to be provided once during a single dialog, the server MUST remember all attributes during the dialog.

When a SAML Assertion contains a SubjectConformation element using the KeyInfoConfirmationDataType as a subject conformation element, the confirmation shall be performed by the creation of an XML Signature authentication element. The signature element shall be created using an appropriate algorithm for the key referenced in the SAML statement.

Identify a SAML statement in the delegation/subject/environment space - need text for this [[CREF6: I don't remember what this is supposed to be anymore. --JLS]]

5.1.2. WS Trust Tokens

WS Trust tokens are used in two different ways by this specification. They can be used as the primary introduction method of a client to the server, or they can be used by the server to allow the client to be re-introduced to the server in such a way that the server can use cached information.

WS Trust tokens come in two basic flavors: Bearer tokens and Holder of Key tokens. With the first flavor, presentation of the token is considered to be sufficient to allow the server to validate the identity of the presenter and know the appropriate attributes to make a policy decision. In the second flavor some type of cryptographic operation (usually a signature or MAC computation) is needed in addition to just presenting the token. The Signature element (Section 5.1.3) provides necessary infrastructure to permit the cryptographic result to be passed to the server.

This document does not define the content or structure of any tokens to be used. This is strictly an implementation issue for the servers in question. This is because the client can treat the WS Token value presented to it as an opaque blob.[[CREF7: Is this totally true? Don't we need some kind of identifier so the server can indicate when the token can be replayed in a subsequence request? E.g. give me these attributes or a foo token. --trevor]] Only the servers need to understand how to process the blob. However there are some additional fields which can be returned in addition to the token that need to be discussed:

wst:TokenType SHOULD be returned if more than one type of token is used by the set of servers. If a token type is returned to the client, the client MUST include the element when the token is returned to the server.

wst:BinarySecret SHOULD be returned for moderate duration tokens. If a binary secret is returned then the client MUST provide protection for the secret value. When a binary secret has been returned, then the client MUST create either a signature or MAC value and place it into the Signature element Section 5.1.3. [[CREF8: I don't know of any way to say use the asymmetric key that you authenticated with originally - can this be done? --JLS]].

wst:Lifetime MUST be returned with the wsu:Expires element set. The wsu:Created element MAY be included. The element provides the client a way to know when a token is going to expire and obtain a new one as needed.

5.1.3. XML Signature Element

When a holder of key credential is used to determine the attributes associated with an entity, there is a requirement that the key be used in a proof of possession step so that the Plasma server can validate that the entity does hold the key. The credentials can hold either asymmetric keys (X.509 certificates and SAML Assertions) or symmetric keys (WS Trust Tokens and SAML Assertions) which use Digital Signatures or Message Authentication Codes (MACs) respectively to create and validate a key usage statement. The XML signature standard [XML-Signature] provides an infrastructure to for conveying the proof of possession information.

The signature is computed over the XACML request element as a detached signature. When a signature element exists in the message, the ChannelBinding attribute (Section 10.1.1) MUST be included in the request. By the use of a value which is derived from the cryptographic keys used in for protecting the tunnel, it is possible for the server to verify that the authentication values computed were done specifically for this specific dialog and are not replayed.

When creating either a signature or a MAC, the following statements hold:

- The canonicalization algorithm Canonical XML 1.1 [XML-C14N11] without comments MUST be supported and SHOULD be used in preparing the XML node set for hashing. Other canonicalization algorithms MAY be used.

- The signature algorithms RSAwithSHA256 and ECDSAwithSHA256 MUST be supported by servers. At least one of the algorithms MUST be supported by clients. The MAC algorithm HMAC-SHA256 MUST be supported by both clients and servers. Other signature and MAC algorithms MAY be supported.

- Set the additional attributes that must be included in a signature - what should they be?

- If an xacml:Request element is referenced by an XML Signature element, the xacml:Request element MUST include the ChannelBinding token (Section 10.1.1) as one of the attributes.

- The keys used in computing the authentication value need to be identified for the recipient. For X509 certificates, the full raw certificate will normally be included as part of the signature, but MAY be referenced instead. For SAML assertions, the specific assertion carrying the asymmetric key can be identified by TBD HERE. In the event that symmetric keys are used by holder of key assertions, the specific assertion will be identified by TBD HERE. In these cases the server is expected to be able to associated the key with the assertion by some means (either locally or perhaps encrypted into the assertion).

5.1.4. GSS-API Element

GSS-API [RFC2743] provides a security services to callers in a generic fasion, supportable with a range of underlying mechanisms and technologies. GSS-API has been extended by providing a mechanism for EAP [RFC7055] which is designed to work in a federated environment. This effort was done by the Application Bridging for Federated Access Beyond web (ABFAB) working group. In this document the mechanism is referred to as ABFAB. This is the same type of environment that the Plasma protocol is expected to operate as well.

GSS-API offers a number of security services that are not currently used by the Plasma system. At this point in time we are only looking at the initial authentiction methods and not using the message encryption or encryption services.

TBD - rules for using GSS-API in general and the EAP version from ABFAB particularly.

- How to build the name.
- Must use a secure tunnel for the outer EAP method and an appropriate inner EAP method(s) to accomplish the required level of authentication.
- Server query of attributes and specification of LOA to the EAP IdP.
- Any additional Trust model items.
- How round trips are accomplished - the only case that a server will send back an Authentication element is on return processing of GSS-API messages.

5.1.4.1. Generic Requirements

Not all GSS-API mechanisms have the required features to support the necessary security that is needed by Plasma. GSS-API mechanisms need to support the following features:

- The mechanism MUST support the binding of the TLS tunnel to the authentication via channel binding.

- Either the mechanism MUST support mutual authentication or the TLS tunnel MUST be usable to authenticate the server being talked to. Anonymous TLS sessions can be use when mutual authentication is provided by the GSS-API mechanism.

5.1.4.2. GSS-EAP Requirements

When forming a mechnism name for GSS-API the following guidelines SHOULD be followed:

- The user-or-service string is "plasma" (all in lower case).

- The host name is to be provided. When obtained from the URL, the host name is to be the entire host name in the URL.

- There are currently no service-specific options defined.

- The realm name is OPTIONAL. When obtained from the URL, the realm name is omitted.

- Realm and host names should be prepared according to [RFC5891] prior to passing them into GSS-API.

Clients MUST use a tunneling EAP method that supports channel binding between the tunnel and the inner EAP methods. At this point in time only the TEAP method [I-D.ietf-emu-eap-tunnel-method] provides the necessary support. While any inner EAP method can be used, it is strongly recommended that only those methods that support generation of EMSK (extended master session keys) be used, however methods tht only support generation of a MSK (master session key) can be used. (A discussion of why EMSKs should be generated can be found in [RFC7029].)

IdPs MUST support the EAP channel binding that is part of TEAP. At a minimum the service name, host name and real names MUST be checked for matches between the information provided by the TEAP channel binding and the RADIUS attributes.

5.1.4.3. GSS-API Channel Bindings

The calls to GSS_Init_sec_content and GSS_Accept_sec_context take a chan_bindings parameter. The value is a GSS_CHANNEL_BINDINGS structure [RFC5554].

The initiator-address-type and acceptor-address-type fields of the GSS-CHANNEL-BINDINGS structure MUST be set to 0. The initiator-address and acceptor-address fields MUST be the empty string.

The application-data field MUST be set to the channel binding value defined in Section 10.1.1.

5.2. xacml:Request Element

The request for an action to be performed by the Plasma server along with the data that needs to be supplied by the client in order for the server to complete the action are placed into the xacml:Request element of the request. This document defines a set of actions that are to be understood by the Plasma server. One (or more) action is to be placed in the request message.

In addition to the request for a specific action to occur, the client can place additional attributes in the request as well. These attributes are provided in order to assist the server either in identifying who the various agents on the client side are or to provide suggestions of attributes for using in making control decisions. Any data provided by the client in this manner is to be considered as a self-asserted value and to be treated as if it comes from the client as oppose to a trusted attribute agent.

For convenience the schema for the xacml:Request element is reproduced here:

```
<xs:element name="Request" type="xacml:RequestType"/>
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element ref="xacml:RequestDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Attributes" maxOccurs="unbounded"/>
    <xs:element ref="xacml:MultiRequests" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="ReturnPolicyIdList" type="xs:boolean" use="required"/>
  <xs:attribute name="CombinedDecision" type="xs:boolean" use="required"/>
</xs:complexType>
```

The RequestDefaults element of the XACML Request MUST be omitted by the clients. If present servers MUST ignore the RequestDefaults

element. The use of the MultiRequest element is current not defined for a Plasma server and SHOULD be omitted by clients.

Clients MAY set ReturnPolicyIdList to true in order to find out which policies where used by the server in making the decision. Server MAY ignore this field and not return the policy list even if requested.

A number of different entities may need to be identified to Plasma server as part of a request. These entities include:

1. The subject making the request to the server.

2. The machine on the subject is using.

3. The entity the subject is acting for. Converse about Delegation.

6. Plasma Response Element

There is a single top level structure that is used by the server to respond to a client request.

The XML Schema used to describe the top level response is as follows:

```
  <xs:element name="PlasmaResponse" type="eps:ResponseType"/>
  <xs:complexType name="ResponseType">
    <xs:sequence>
      <xs:element ref="xacml:Response"/>
      <xs:element ref="eps:PlasmaReturnToken" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Version" type="xs:string" default="1.0"/>
  </xs:complexType>
  <xs:element name="PlasmaReturnToken" type="eps:PlasmaReturnTokenType"/>
  <xs:complexType name="PlasmaReturnTokenType">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax"/>
    </xs:sequence>
    <xs:attribute name="DecisionId" type="xs:string"/>
  </xs:complexType>
```

A Plasma Response has two elements:

xacml:Response is a mandatory element that returns the status of the access request.

PlasmaReturnToken is an optional element to return a token. These tokens represent the answer, for a success, of the request. If multiple tokens are being returned, then the element will occur mutiple times.

A Plasma Return Token is a wrapper for the actual token being returned. The returned token may be any content. This document defines the following elements that are to be returned in this location

- RoleToken - used to return roles.
- CMSMessageToken - used to return one or more CMS RecipientInfo strucutures.
- CMSKeyToken - used to return either a CMS RecipientInfo structure or a bare content encryption key.

The PlasmaReturneTokenType has an optional attribute DecisionId. This attribute is used when in the case multiple requests are made at the same time in order to assoicate the rquest and the response tokens.

6.1. xacml:Response Element

The xacml:Response element has the ability to return both a decision, but additionally information about why a decision was not made.

The schema for the xacml:Response element is reproduced here for convenience:

```
<xs:element name="Response" type="xacml:ResponseType"/>
<xs:complexType name="ResponseType">
  <xs:sequence>
    <xs:element ref="xacml:Result" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="Result" type="xacml:ResultType"/>
<xs:complexType name="ResultType">
  <xs:sequence>
    <xs:element ref="xacml:Decision"/>
    <xs:element ref="xacml:Status" minOccurs="0"/>
    <xs:element ref="xacml:Obligations" minOccurs="0"/>
    <xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
    <xs:element ref="xacml:Attributes" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:PolicyIdentifierList" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The xacml:Response element consists of one child the Result.

The xacml:Response element consists of the following elements:

xacml:Decision is a mandatory element that returns the possible decisions of the access control decision. The set of permitted values are Permit, Deny, Indeterminate and No Policy.

xacml:Status is an optional element returned for the Indeterminate status which provides for the reason that a decision was not able to be reached. Additionally it can contain hints for remedying the situation. This document defines an additional set of status values to be returned. Formal declaration may be found in Section 15.

* gss-api indicates that a gss-api message has been returned as part of the authentication process.

xacml:Obligations is designed to force the PEP to perform specific actions prior to allowing access to the resource. If a response is returned with this element present, the processing MUST fail unless the PEP can perform the required action. A set of Plasma specific obligations are found in Section 10.2. [[CREF9: What about audit obligatiouns --Trevor]]

xacml:AssocatedAdvice is designed to give suggestions to the PEP about performing specific actions prior to allowing access to the resource. This element is not used by Plasma and SHOULD be absent. If the response is returned with this element present, processing will succeed even if the PEP does not know how to perform the required action. A set of Plasma specific advice elements are found in Section 10.2.

xacml:Attributes provides a location for the server to return attributes used in the access control evaluation process. Only those attributes requested in the Attributes section of the request are to be returned. Since Plasma does not generally supply attributes for the evaluation process, this field will normally be absent.

xacml:PolicyIdentifierList provides a location to return the set of policies used to grant access to the resource. This element is expected to be absent for Plasma. [[CREF10: Should we ignore it if present? --Trevor]][[CREF11: I don't think we need to say anything about looking at it or ignoring it. While it would be something for debugging, as a general rule the client does not care which policies where evaluated and passed to grant access. --JLS]]

7. Role Token and Policy Acquisition

In order to send an email using a Plasma server, the first step is to obtain a role token that provides the description of the labels that can be applied and the authorization to send an email using one or more of the labels. The process of obtaining the role token is designed to be a request/response round trip to the Plasma server. In practice a number of round trips may be necessary in order to provide all of the identity and attributes to the Plasma server that are needed to evaluate the policies for the labels.

When a Plasma server receives a role token request from a client, it needs to perform a policy evaluation for all of the policies that it arbitrates along with all of the options for those policies. In general, the first time that a client requests a role token from the server, it will not know the level of authentication that is needed or the set of attributes that needs to be presented in order to get the set of tokens. A server MUST NOT issue a role token without first attempting to retrieve from an attribute source (either the client or a back end server) all of the attributes required to check all policies. Since the work load required on the server is expected to be potentially extensive for creating the role token, it is expected that the token returned will be valid for a period of time. This will allow for the frequency of the operation to be reduced. While the use of an extant role token can be used for identity proof, it is not generally suggested that a new token be issued without doing a full evaluation of the attributes of the client as either the policy or the set of client attributes may have changed in the mean time.

7.1. Role Token Request

The process starts by a client sending a server a role token request. Generally, but not always, the request will include some type of identity proof information and a set of attributes. It is suggested that, after the first successful conversation, the client cache hints about the identity and attributes needed for a server. This allows for fewer round trips in later conversations. An example of a request token can be found in Appendix B.

The role token request, as with all requests, uses the eps:PlasmaRequest XML structure. The eps:Authentication MAY be included on the first message and MUST be included on subsequent authentication round trips.

A role token request by a client MUST include the GetRoleTokens Plasma action request as an attribute of the xacml:Request element. Details on the action can be found in section Section 15.1. When

role tokens are requested, no additional data needs to be supplied by the requester.

An example of a message requesting the set of policy information is:

```
<esp:PlasmaRequest>
  <eps:Authentication>...</eps:Authentication>
  <xacml:Request>
    <xacml:Attributes Category="...:action">
      <xacml:Attribute AttributeId="urn:plasma:action-id">
        <xacml:AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#string">
          GetRoleToken</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
  </xacml:Request>
</esp:PlasmaRequest>
```

7.2. Request Role Token Response

In response to a role token request, the Plasma server returns a role token response. The response uses the eps:PlasmaResponse XML structure. When a response is create the following should be noted:

An xacml:Decision is always included in a response. The values permitted are:

Permit is used to signal success. In this case the response MUST include one or more eps:RoleToken element.

Deny is used to signal failure. In this case the xacml:Status element MUST be present an contain a failure reason.

Indeterminate is used to signal that a final result has not yet been reached. When this decision is reached, the server SHOULD return a list of additional attributes to be returned and SHOULD return the list of role tokens that have been granted based on the attributes received to that point.

NotApplicable is returned if the Plasma server does not have the capability to issue role tokens.

An example of a response returning the set of policy information is:

```
<eps:PlasmaResponse>
  <xacml:Response>
    <xacml:Result>
      <xacml:Decision>Permit</xacml:Decision>
    </xacml:Result>
  </xacml:Response>
  <eps:PlasmaTokens>
    <eps:PlasmaToken>
      <eps:PolicyList>
        <eps:Policy>
          Details of a policy
        </eps:Policy>
        ... More policies ...
        <wst:RequestSecurityTokenResponse>
          <wst:TokenType>urn:...:plasma:roleToken</wst:TokenType>
          <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
        </wst:RequestSecurityTokenResponse>
      </eps:PolicyList>
    </eps:PlasmaToken>
  </eps:PlasmaTokens>
</eps:PlasmaResponse>
```

The process of getting role tokens has a problem that is not part of the normal XACML design. It is possible to compute a partial result based on the current set of attributes that have been supplied by the client, while having other role tokens that cannot be provided to the client since required attributes have not been provided. Since this is not part of the standard XACML model, one only has a single access /deny decision and if the attributes have not been provided then the response would be deny, we need to look at it in a bit more detail here.

In the process of discussions three different solutions to the problem were considered:

A signal could be added that allows for the client to signal that it cannot provide any more attributes to the server. This would allow for a final decision to be provided, but would potentially involve an additional round trip as the set of attributes can be determined based on the set of attributes provided. Supplying new attributes from the client can result in the server asking for new attributes from the client. This is not currently supported by the XACML model and there is no clear point where it would go into our model.

The server can return a partial result on each round trip. This maps directly onto the XACML model, but leads to some other problems. It means that all of the policies must be designed such

that adding a new attribute to the policy evaluation process will not cause a policy that previously had been permitted is now denied.

A method could be added that allows for the client to state that it either does not have or does not know what an attribute is. This method would allow for the server to make a definitive answer, but it requires that one extra round trip be made to get the final answer.

The normal mode that Plasma servers are expected to operate in is returning incremental results, however they can also keep internal state looking at what additional attributes are being provided by the client. If the client provides no new attributes, then the server can return a set of role tokens close down the conversation. If the server expects to get all attributes from the back end, and just get authentication from client, then it can return a set of role tokens immediately without providing a list of attributes to the client for it to try and satisfy.

7.2.1. RoleToken XML element

The eps:PlasmaReturnToken element is used to return a role token to the client. Multiple role tokens can be returned by using multiple eps:PlasmaReturnToken elements. Each role token returned contains one or more policies that can be asserted, the role token, and optionally one or more set of obligations or advice that need to be observed when creating messages. Additionally the name of a Plasma server to be used with the token can be included as well as cryptographic information to be used with the token.

The schema used for the PlasmaTokens element is:

```
  <xs:element name="RoleToken" type="eps:RoleTokenType"/>
  <xs:complexType name="RoleTokenType">
    <xs:sequence>
      <xs:element name="FriendlyName" type="xs:string"/>
      <xs:element name="PDP" type="xs:anyURI" maxOccurs="unbounded"/>
      <xs:choice>
        <xs:element name="PolicyList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Policy" type="eps:PolicyDescType"
maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element ref="eps:Policy"/>
        <xs:element ref="eps:PolicySet"/>
      </xs:choice>
      <xs:element ref="wst:RequestSecurityTokenResponse"/>
      <xs:element ref="xacml:Obligations" minOccurs="0"/>
      <xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="PolicyDescType">
    <xs:sequence>
      <xs:element name="FriendlyName" type="xs:string"/>
      <xs:element name="Options" minOccurs="0">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="xs:anyType">
              <xs:attribute name="optionsType" type="xs:anyURI" use="required"/
>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
  </xs:complexType>
```

The eps:RoleToken element contains the following items:

FriendlyName This element returns a descriptive name of the role as a whole. The string returned SHOULD be selected based on the language attribute on the request message. The string is suitable for display to the user and should be indicative of the scope of the role. Examples of role descriptive strings would be "Company President", "Senior Executive", "Project X Electrical Engineer".

PDP The element provides one or more URLs to be used for contacting
   a Plasma server for the purpose of sending a message. This

element allows for the use of different Plasma servers for issuing role tokens and message tokens. No ranking of the servers is implied by the order of the URLs returned.[[CREF12: Should perhaps rename to be more understandable - perhaps Server --JLS]]

PolicyList contains the description of one or more policies that can be asserted using the issued role token. Any of the policies contained in the list may be combined together using the policy logic in constructing a label during the send message process.

Policy contains a single simple policy. This element is returned as part of a read message token. The purposes is to allow for a recipient to reply to a message that they would not normally be able to assert.

PolicySet contains a complex policy. This element is returned as part of a read message token The purpose is to allow for a recipient to reply to a message that they would not normally be able to assert.

wst:RequestSecurityTokenResponse contains the actual token itself.

xacml:Obligations This optional element contains a set of obligations that the client is required to enforce in order to use any of the policies listed when combined with the returned security token. These obligations can include items such as required algorithms or required operational steps such as requiring a signature to be placed on the content. A policy can be listed in multiple role tokens and the set of obligations may be different depending on which role token is used. If the client is unable to fulfill the obligations then it MUST NOT allow the role token to be used.

xacml:AssociatedAdvice This optional element contains a set of advice statements that the client is requested to enforce when using any of the policies listed when combined with the returned security token. This advice can include items such as encryption or signature algorithms or operational steps such as requiring a signature to be placed on the content. The client is SHOULD fulfill the advice, however if it cannot fulfill the advice the role token may still be used.

The eps:PolicyType type is used to represent the elements of a policy to the client. The elements in this type are:

FriendlyName contains a display string that represents the policy. This element is localized in response to the xs:lang attribute in the eps:PlasmaRequest node.

PolicyId contains a "unique" identifier for the policy. This is the value that identifies the policy to the software. The type for the value is defined as a URI.

Options This element is used to inform the client what the set of options that need to be filled in as part of asserting the policy. If the client software does not understand how to set the options for the supplied type, then the client software MUST NOT allow the user to assert the policy. The option structure is identified by the URI in the optionsType attribute. This document defines one option structure for holding a list of email addresses (section Section 7.2.2). This option structure is used in the basic policies defined in [PlasmaBasicPolicy].

When building the wst:RequestSecurityTokenResponse element, the following should be noted:

A wst:RequestedSecurityToken element containing the security token MUST be included. The format of the security token is not specified and is implementation specific, it is not expected that clients should be able to get useful information from the token itself. Information such as lifetimes need to be provided as addition elements in the response. Examples of items that could be used as security tokens are SAML statements or encrypted record numbers in a server database.

A wst:Lifetime giving the life time of the token SHOULD be included. It is not expected that this should be determinable from the token itself and thus must be independently provided. There is no guarantee that the token will be good during the lifetime as it may get revoked due to changes in the environment (for example, if the policies are updated then all existing tokens may need to be re-issued), however the client is permitted to act as if it were. The token provided may be used for duration. If this element is absent, it should be assumed that the token is either a one time token or of limited duration.

Talk about cryptographic information - There are three different types of crypto information that can be returned and we need to figure out how to talk about them. These are 1) a symmetric key, 2) a new asymmetric key and 3) a pre-existing asymmetric key - for example from the certificate used for authentication purposes. There is probably good ways to do 1 and 2, but I don't know how to talk about 3 at this point in time.

7.2.2. Email Address List Options

Some policies are designed to be restricted to a set of explicitly named people by the sender of the message. This policy is used for the set of basic policies defined in [PlasmaBasicPolicy]. In these cases the creator of the message specifies a set of recipients by using email address names without any decoration.

The Email Address List Option is identified by the uri "urn:ietf:params:xml:ns:plasma:options:emailAddrs". The type associated with the structure is a string. The string contains a space separated set of internalized email addresses. Domains SHOULD be encoded as U-labels rather than using puny code.

All Plasma clients and servers MUST be able to create, parse and use the Email Address List Option for any policy.

As of the release of this document, Plasma clients and servers are not expected to understand any other options.

8. Sending An Email

After having obtained a role token from a Plasma server, the client can then prepare to send an Email by requesting a message token from the Plasma server. As part of the preparatory process, the client will construct the label to be applied to the Email from the set of policies that it can assert, determine the optional elements for those policies which have options, generate the random key encryption key and possible create the key recipient structures for the email. Although this section is written in terms of a CMS Encrypted message, there is nothing to prevent the specification of different formats and still use this same basic protocol. An example of a send mail request token can be found in Appendix D.

8.1. Send Message Request

The send message request is built using the eps:PlasmaRequest XML structure. When building the request, the following applies:

- The eps:Authentication element MUST be included in the initial message. The role token that authorizes the use of the label MUST be included in the initial message. If the role token is dependent on a cryptographic key for authentication, then that authentication MUST be included in the initial message.

- The client MUST include an action attribute. This document defines the GetSendCMSToken action attribute for this purpose.

- The client MUST include a data attribute. This attribute contains the information that is used to build the CMS Message token to be returned. There MAY be more than one data attribute, but this will not be a normal case. More details on this attribute are in Section 8.1.1.

- If the client is using the XML Digital Signature element in this message, then the client MUST include the cryptographic channel binding token (Section 10.1.1) in the set of XACML attributes.

A message requesting that a CMS message token be created looks like this:

```
<eps:PlasmaRequest>
  <eps:Authentication>
    <eps:WS_Token>
      Role Token goes here
    </eps:WS_Token>
    <xacml:Request>
      <xacml:Attributes Category="...:action">
        <xacml:Attribute AttributeId="urn:plasma:action-id">
          <xacml:AttributeValue>
            GetSendCMSToken
          </xacml:AttributeValue>
        </xacml:Attribute>
      </xacml:Attributes>
      <xacml:Attributes Category="...:data">
        <xcaml:Attribute AttributeId="urn:plasma:data-id">
          <xacml:AttributeValue>
            Label and keys
          </xacml:AttributeValue>
        </xcaml:Attribute>
      </xacml:Attributes>
    </xacml:Request>
  </eps:Authentication>
</eps:PlasmaRequest>
```

8.1.1. CMS Message Token Data Structure

The message token data structure is used as an attribute to carry the necessary information to issue a CMS message token. The schema that describes the structure is:

```
  <xs:element name="GetCMSToken" type="eps:CMSTokenRequestType"/>
  <xs:complexType name="CMSTokenRequestType">
    <xs:sequence>
      <xs:choice>
        <xs:element ref="eps:Policy"/>
        <xs:element ref="eps:PolicySet"/>
      </xs:choice>
      <xs:element name="Hash">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="ds2:DigestMethod"/>
            <xs:element ref="ds2:DigestValue" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="LockBox" type="eps:LockBoxType" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="CEK" type="xs:hexBinary" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="LockBox" type="eps:LockBoxType"/>
  <xs:complexType name="LockBoxType">
    <xs:sequence>
      <xs:element name="Subject" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:anySimpleType">
              <xs:attribute name="type" type="xs:string" use="required"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:choice>
        <xs:element name="CMSLockBox" type="xs:base64Binary"/>
        <xs:element name="XMLLockBox" type="xenc:EncryptedKeyType"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
```

When used in an xacml:Attribute, the structure is identified by:

Category = "urn:ietf:params:xml:ns:plasma:data"
AttributeId = "urn:ietf:params:xml:ns:plasma:data:CMSTokenRequest"
DataType =
"urn:ietf:params:xml:schema:plasma:1.0#CMSTokenRequestType"

The elements of the structure are used as:

Policy
This element contains a the policy to be applied to the message when a single policy is used.

PolicySet
This element contains the policy to be applied to the message when a combination of policies is to be applied.

Hash
This element contains the hash of the encrypted content of the message that the policy is being applied to. The algorithm used to compute the hash is contained in the DigestMethod element and the value is contained in the DigestValue element.

LockBox
This optional element contains a pre-computed CMS recipient info structure for a message recipient. This element may be repeated when more than one lock box is pre-computed for recipients by the message sender. This element is used in those cases where the sender does not want to share the content encryption key with the Plasma server and the sender has the ability to retrieve the necessary keys for all of the recipients. If the #### obligation was returned for the role token, then a recipient info lock box MUST be created for the Plasma server and the CEK element MUST absent. [[CREF13: Do we define this obligation or remove the previous sentence? --JLS]]

CEK
This optional element contains the content encryption key (CEK) to decrypt the message.

One or both of CEK and Recipients elements MUST be present.

The elements of the LockBoxType structure are:

Subject
This element contains a subject identifier. The element can occur more than one time in situations where a subject has multiple names or a key is used by multiple subjects. Since a CMS recipient info structure does not contain a great deal of information about the recipient, this element contains a string which can be used to identify the subject. The format of the subject name is provided by the required type attribute of the element. All implementations MUST recognize "urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress" as a name type. [[CREF14: Call for other mandatory to implement name types --JLS]] Other name types MAY be recognized.

CMSLockBox
This element contains a base64 encoded CMS Recipient Info structure. If the recipient info structure is placed here, it MUST NOT be placed in the CMS EnvelopedData structure as well.

XMLLockBox
This element contains an EncryptedKeyType structure as defined by the XML Encryption standard [W3C.WD-xmlenc-core1-20101130]. If this recipien structure is placed here, it MUST NOT placed in the XML EncryptedType struture as well.

In addition, the structure allows for other formats of encrypted data structures to be included as well. Servers which do not recognize the name space and data structure MUST return an unrecognized data structure error and not process the request.

8.1.2. XML Label Structure

A client is allowed to build a complex label to be sent to the Plasma server for evaluation. While there are some cases that a simple single policy is applied to a message, it is expected that many, if not most, messages will have more than one policy applied to it with logical statements connected those policies.

The schema for specifying a label is:

```
<xs:element name="PolicySet" type="eps:PolicySetType"/>
<xs:complexType name="PolicySetType">
  <xs:sequence>
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="eps:Policy"/>
      <xs:element ref="eps:PolicySet"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>
</xs:complexType>
<xs:element name="Policy" type="eps:PolicyType"/>
<xs:complexType name="PolicyType">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The Policy and the PolicySet elements are used when specifying a policy for a message depending on whether a single policy or multiple policies are to be evaluated.

The Policy element is used to specify a single policy to the server along with any options that are defined for that policy. The Policy element contains:

PolicyId
: Is an attribute that contains the URI which identifies a specific policy to be evaluated.

inner content
: The content of the Policy element can be any XML element. The content is to be the set of selected options for the policy (if any exist). The schema applied to the content is based on the policy selected.

The PolicySet element is used to specify a logical set of policies to be applied to the message. This element allows one to specify multiple policies along with a logic operation to combine them together.

Policy
: This element allows for a single policy and any policy specific options for the policy to be specified. This element can occur zero or more times.

PolicySet
: This element allows for a logical set of policies to be recursively evaluated. This element can occur zero or more times.

PolicyCombiningAlgId
: This attribute specifies the operation to be used in combining the elements of the tree together. This specification uses the XACML policy combining algorithms from [XACML]. Servers and clients MUST support the unordered Deny-Overrides and Permit-Overrides policy combining rules. Servers SHOULD support all of the policy combining rules defined in [XACML]. Clients are expected to use a friendly name when displaying the policy combining rule to users. When displaying strings to users, the following strings are suggested:

  AND Is used to represent either the ordered or unordered Deny-Overrides combining algorithm.

  OR Is used to represent either the ordered or unordered Permit-Overrides combining algorithm.

8.2. Send Message Response

In response to a send message request, the Plasma server returns a send message response message. The response messages uses the eps:PlasmaResponse XML structure. When the response message is created, the following should be noted:

- The xacml:Decision is always included in the response. If the 'Permit' value is returned then a CMS Token Response element MUST be present.

- The PlasmaReturnToken element with a eps:CMSToken content is included with a permit response. The CMSToken contains one or more CMS RecipientInfo objects to be included in the message sent.

An example of a message returning the set of policy information is:

```
<eps:PlasmaResponse>
  <xacml:Response>
    <xacml:Result>
      <xacml:Decision>Permit</xacml:Decision>
    </xacml:Result>
  </xacml:Response>
  <eps:CMSToken>234e34d3</eps:CMSToken>
</eps:PlasmaResponse>
```

The schema use for returning a CMS token is:

```
  <xs:element name="CMSToken" type="eps:CMSTokenResponseType"/>
  <xs:complexType name="CMSTokenResponseType">
    <xs:sequence>
      <xs:element name="CMSLockBox" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:base64Binary">
              <xs:attribute name="CMSType" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

This schema fragment extends the Plasma response token type and allows for the return of one or more base64 encoded RecipientInfo structures. The Plasma server can return recipient info information for any recipient that it pre-authorizes to receive the message (see Section ### of [I-D.freeman-plasma-requirements] for examples of when

this would occur). Additionally the Plasma server can return a KEKRecipientInfo structure with the Plasma Other Key attribute. (For details see [I-D.schaad-plasma-cms].) In some extremely rare cases where the Plasma server can pre-authorize the entire set of recipients , the KEKRecipientInfo structure with the Plasma Other Key Attribute may not be included in the returned set of recipients. The recipient info structure for the plasma server SHOULD be placed last in the list of recipients infos.

The CMSTokenResponse type has the following:

CMSLockBox
: This element contains the ASN.1 encoding for a CMS RecipientInfo structure to be placed in the final message. This element will occur multiple times if there are multiple CMS RecipientInfo structures being returned from the server.

CMSType
: This attribute of the RecipientInfo structure is an optional text value that identifies the type of recipient info structure returned. NOTE: This attribute is currently optional and is likely to disappear if I do not find it useful.

8.3. XML Message Send Request

It is possible to do a send message request for an XML rather than a CMS message structure. The send message request is built using the eps:PlasmaRequest XML structure. When building the request, the following applies:

- The eps:Authentication element MUST be included in the initial message. The role token that authorizes the use of the label MUST be included in the initial message. If the role token is dependent on a cryptographic key for authentication, then that authentication MUST be included in the initial message.

- The client MUST include an action attribute. This document defines the GetSendXMLToken action attribute for this purpose.

- The client MUST include a data attribute. This attribute contains the information that is used to build the XML Message token to be returned. There MAY be more than one data attribute but that is not a normal case. More details on this attribute are in Section 8.1.1.

- If the client using the XML Digital Signature element in this message, then the client MUST include the cryptographic channel binding token (Section 10.1.1) in the set of the XACML attributes.

8.4. XML Message Send Response

In response to a send message request, the Plasma server returns a send message response message. The response messages uses the eps:PlasmaResponse XML structure. When the response message is created, the following should be noted:

- The xacml:Decision is always included in the response. If the 'Permit' value is returned then a XML Token Response element MUST be present.

- The PlasmaReturnToken element with a eps:XMLToken content is included with a permit response. The XMLToken contains one or more XML EncyptedKey objects to be included in the message sent.

```
      <xs:element name="XMLToken" type="eps:XMLTokenResponseType"/>
      <xs:complexType name="XMLTokenResponseType">
        <xs:sequence>
          <xs:element name="XMLLockBox" maxOccurs="unbounded"
type="xenc:EncryptedKeyType"/>
        </xs:sequence>
      </xs:complexType>
```

9. Decoding A Message

When the receiving agent is ready to decrypt the email, it identifies that there is a KEKRecipientInfo object which contains a key attribute identified by id-keyatt-eps-token. It validates the signature, determines that communicating with the Plasma Service is within local policy, and then sends a request to the service to obtain the decryption key for the message.

In some cases the recipient of a message is not authorized to use the same set of labels for sending a message. For this purpose a token can be returned in the message along with the key so that recipient of the can reply to the message using the same set of security labels.

9.1. Requesting Message Key

The client sends a request to the Plasma server that is identified in the token. For the CMS base tokens, the address of the Plasma server to use is defined in [I-D.schaad-plasma-cms] this is located in the aa-eps-url attribute.

The request uses the eps:PlasmaRequest XML structure. When building the request, the following should be noted:

- The xacml:Request MUST be present in the first message of the exchange.

- The action used to denote that a CMS token should be decrypted is "ParseCMSToken".

- The CMS token to be cracked is identified by "CMSToken"

- In the event that a reply to role token is wanted as well, then that is supplied as a separate action. [[CREF15: We may want to require that a reply token always be returned instead of just returning it on demand. --jls]] In this case the action is "GetReplyToken".

- If the client is using the XML Digital Signature element in this message, then the client MUST include the cryptographic channel binding token (Section 10.1.1) in the set of XACML attributes.

An example of a message returning the set of policy information is:

```
<eps:PlasmaRequest>
  <eps:Authentication>...</eps:Authentication>
  <xacml:Request>
    <xacml:Attributes Category="...:action">
      <xacml:Attribute AttributeId="..:action-id">
        <xacml:AttributeValue>ParseCMSToken</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Attribute Category="...:data">
      <xacml:Attribute AttreibuteId="..:data-id">
        <xacml:AttributeValue>
          Hex encoded CMS Token Value
        </xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attribute>
  </xacml:Request>
</eps:PlasmaRequest>
```

When used in an xacml:Attribute, the structure is identified by:

```
Category = "urn:ietf:params:xml:ns:plasma:data"
AttributeId = "urn:ietf:params:xml:ns:plasma:data:CMSToken"
DataType =
"urn:ietf:params:xml:schema:plasma:1.0#CMSTokenResponseType
```

9.2. Requesting Message Key Response

In response to a message key request, the Plasma server returns a decrypted key in the message key response. The response message uses the eps:Plasma XML structure. When a response message is create the following should be noted:

- If the value of xacml:Decision is Permit, then response MUST include an eps:CMSKey element.
- For all other decision types the eps:CMSKey MUST be absent.
- If a reply token was requested and granted, then the response MUST include an eps:PlasmaToken element. The eps:PlasmaToken element MUST use the Label option
- Only the CEK and CMSLockBox elements in the choice are permitted for the CMSKey tag name.

An example of a message returning the set of policy information is:

```
<eps:PlasmaResponse>
  <xacml:Response>
    <xacml:Result>
      <xacml:Decision>Permit</xacml:Decision>
    </xacml:Result>
  </xacml:Response>
  <eps:CMSKey>
    <eps:DisplayString>Label TExt</eps:DisplayString>
    <eps:KEK>hex based KEK</eps:KEK>
  </eps:CMSKey>
</eps:PlasmaResponse>
```

The schema for returning the decrypted key is:

```
  <xs:element name="CMSKey" type="eps:CMSKeyResponseType"/>
  <xs:complexType name="CMSKeyResponseType">
    <xs:sequence>
      <xs:element name="DisplayString" type="xs:string"/>
      <xs:choice>
        <xs:element name="CEK" type="xs:base64Binary"/>
        <xs:element name="CMSLockBox" type="xs:base64Binary"/>
                                <xs:element name="XMLLockBox"
type="enc:EncryptedKeyType"/>
                                <xs:any namespace="##other"
processContents="lax"/>
      </xs:choice>
      <xs:element ref="eps:RoleToken" minOccurs="0"/>
      <xs:element ref="xacml:Attributes" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

This schema extends the Plasma response token type and restricts the content to the listed elements. The values returned are:

DisplayString returns a localized display string for the policy(s) which were applied to the message. The lang attribute on the request is used to determine which language to use for this string.

CEK returns the base64 encoded content encryption key.

CMSLockBox returns the content encryption key in the form of a CMS RecipientInfo structure.

XMLLockBox returns the content encryption key in the form of an XML Encrypted Key type.

RoleToken optionally returns a role token for replying to this message.

Attributes optionally returns a set of attributes associated with the message.

The structure allows for additional key types to be defined in other schemas and returned in this structure as well. The set of allows lock boxes to be returned is restricted by the XML tag nd not the schema.

10. Plasma Attributes

In this document a number of different XACML attributes have been defined, this section provides a more detailed description of these elements.

10.1. Data Attributes

10.1.1. Channel Binding Data Attribute

The channel binding data attribute is used to provide for a binding of the TLS session that is being used to transport the Plasma messages with the content of the Plasma requests themselves. There is a need for the server to be able to validate that the cryptographic operations related to holder of key statements be made specifically for the current conversation and not be left over from a previous one as a replay attack. By deriving a cryptographic value from the shared TLS session key and signing that value we are able to do so.

The channel binding value to be used is created by the TLS key exporter specification defined in RFC 5705 [RFC5705]. This allows for a new cryptographic value to be derived from the existing shared secret key with additional input to defined the context in which the key is being derived. When using the exporter, the label to be input into the key exporter is "EXPORTER_PLASMA". The value to be derived is 512 bits in length, and no context is provided to the exporter.

When used as an XACML attribute in a request:

The category of the attribute is "urn:ietf:params:xml:ns:plasma:data".

The AttributeId attribute is "urn:ietf:params:xml:ns:plasma:data:ChannelBinding".

The Issuer attribute is absent.

The DataType is either base64Binary or hexBinary

The same value is used for both the XACML channel binding data attribute and the XM1L channel binding structure defined in Section 5.1.3.

10.1.2. CMS Signer Info Data Attribute

In many cases a policy states that the client is required to sign the message before encrypting it. The server cannot verify that a signature is applied to the message and included, but we can require that a signature be supplied to the server. This signature can then be validated by the server (except for the message digest attribute value), and the server can take a hash of the value and return it as part of the key returned to a decrypting agent. This agent can then validate that the signature is a part of the message and complain if

it absent. This means we do not have an enforcement mechanism, but we do have a way of performing an audit at a later time to see that the signature operation was carried out correctly.

By requiring that a signature be supplied to the server as part of the authentication process, the Plasma server can also be setup so that the supplied signature is automatically feed into archival operations. One way to do archiving is to use the data records defined in [RFC4998].

The following applies when this data value is present:

The Category attribute is "urn:ietf:params:xml:ns:plasma:data".

The AttributeId attribute is "urn:ietf:params:xml:ns:plasma:data:CMSSignerInfo".

The Issuer attribute is absent.

The DataType attribute is either base64Binary or hexBinary.

The data value is a CMSSignerInfo ASN.1 encoded object.

10.1.3. S/MIME Capabilities Data Attribute

Policies sometimes require that specific algorithms be used in order to meet the security needs of the policy. This attribute allows for an S/MIME Capabilities to be carried in a DER encoded SMIMECapabilities ASN.1 structure to be transmitted to the client. Details on how the S/MIME Capabilities function can be found in [RFC5751].

The following attributes are to be set for the data value:

The Category attribute is "urn:ietf:params:xml:ns:plasma:data".

The AttributeId attribute is "urn:ietf:params:xml:ns:plasma:data :SMIME-Capabilties".

The Issuer attribute is absent.

The DataType attribute is either base64binary or hexBinary.

10.1.4. EMAIL Recipient Addreses

In order for Plasma Servers to do pre-authentication in the Email environment, it is necessary for the set of recipient addresses to be delivered to the Plasma Server. The Plasma Server cannot reliably

determine the set of recipients from the policies set on the message as the set of recipients and the set of people authorized to view the message may not have a one-to-one correspondance. People may be authorized to see the content who are not recipients of the message or visa versa.

The content of the attribute is a space separated list of email addresses. Each address represents an Email recipient address that the client will be placing in one or more of the recipient fields in the message submission.

The following attributes are to be set for the data value:

The Category for the attribute is "urn:ietf:params:xml:ns:plasma:data".

The AttributeId for the attribute is "urn:ietf:params:xml:ns:plasma:data:SMTPRecipients".

The Issuer for the attribute is absent.

The DataType for the attribute is "http://www.w3.org/2001/XMLSchema#string".

10.1.5. Return Lockbox Key Information

Some policies require that the content encryption key be transported wrapped by another key rather than being sent in plain text. This data value allows for this state to be indicated by the Plasma Server to the Plasma Client, and for the client to provide the necessary key information to the server.

This data attribute is returned as a missing attribute under the circumstances where it is required by the policy and has not been provided the client. This is an indication that the content encryption key needs to be returned in a lock box rather than as plain text. The Plasma Server MAY ignore this data value if it is provided in a situation where the policy does not require that the content encryption key be returned in an encrypted form.

The following attributes are to be set for this data value:

The Category for the attribute is "urn:ietf:params:xml:ns:plasma:data".

The AttributeId for the attribute is "urn:ietf:params:xml:ns:plasma:data:LockboxKey".

The issue for the attribute is absent.

The DataType for the attribute is "urn:ietf:params:xml:schema:plasma:1.0LockboxKey".

The schema for the type LockboxKey is:

```
<xs:complexType name="LockboxKey">
  <xs:sequence>
    <xs:choice>
      <xs:element name="X509Certificate" type="xs:base64Binary"/>
      <xs:element name="PGPKey" type="xs:base64Binary"/>
      <xs:element ref="ds2:KeyInfo"/>
    </xs:choice>
    <xs:element name="Capabilities" type="xs:base64Binary" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The fields of this structure are as follows:

X509Certificate holds a certificate with the public key to be used in building the CMS Lockbox returned containing the content encryption key.

PGPKey holds a PGP public key to be used in building the PGP lock box returned containing the content encryption key.

ds2:KeyInfo holds a public key to be used in building a CMS lock box returned containing the content encryption key.

Capabilities contains a base64 encoded SMimeCapablities ASN.1 structure allowing the client to advertise to the server which algorithms are supported for building the lockbox structure. This element is optional. If the element is omitted then the server will make the selection of algorithms to be used based solely on the pubic key.

10.2. Obligations and Advice

Obligations and advice consist of actions that the Plasma server either requires or requests that the client PEP perform in order to gain access or before granting access to the data. These normally represent actions or restrictions that the PDP itself cannot enforce and thus are not input attributes to the policy evaluation. The same set of values can be used either as obligations or advice, the difference being that if the PEP cannot do an obligation it is required to change the policy decision.

10.2.1. Signature Required

Many policies require that a message be signed before it is encrypted and sent. Since the unencrypted version of message is not sent to the Plasma server, the policy cannot verify that a signature has been placed onto the signed message. The attribute is not for use as a returned obligation from an XACML decisions, rather it is for a pre-request obligations used in role responses (Section 7.2).

When used as an Obligation:

The ObligationId attribute is "urn:ietf:params:xml:ns:plasma:obligation:signature-required".

A S/MIME Capabilities data value can optionally be included. If it is included, then it contains the set of S/MIME capabilities that describes the set of signature algorithms from which the signature algorithm for the message MUST be selected.

10.2.2. Content Encryption Algorithm Required

Occasionally a policy requires a specific set of encryption algorithms be used for a message, when this is the case then the encryption required obligation is included in the returned set of obligations. If the default set of encryption algorithms is sufficient then the obligation is omitted.

When used as an Obligation:

The ObligationId attribute is "urn:ietf:params:xml:schema:plasma:1.0:obligation:content-encryption".

An S/MIME Capabilities data value MUST be included containing the set of permitted encryption algorithms. The algorithms included MUST include a sufficient set of algorithms for the message to be encrypted. An absolute minimum would be a content encryption algorithm and key encryption algorithm.

10.2.3. Lock Box Required

This obligation will be used in one of two situations:

1. The policy requires that the plain content encryption key not be given to the Plasma server, but instead the Plasma client is required to locate the appropriate certificates and create lock boxes for each of the message recipients. In this situation, the Plasma server would never have any access to the content

encryption key and thus would be unable to provide the key to any entity. The Plasma server in this case is responsible only for the enforcement of the policy enforcement on the message access.

2. The policy requires that the content encryption key not be given to the Plasma server as a base64 encoded blob, but instead the Plasma client is required to use the provided certificate to create a lock box for the Plasma server. In this situation, the Plasma server does have access to the content encryption key and thus has the ability to do late binding. The Plasma server is also still responsible for the enforcement of the policy on message access.

When used as an Obligation:

The ObligationId attribute is "urn:ietf:params:xml:schema:plasma:1.0:obligation:lockbox-required".

There is no data value when the client is required to create lock boxes for every recipient, i.e. early binding.

The data value is an X509 certificate when the Plasma client is required to create a lock box for the Plasma server. The certificate provided MUST have the Plasma CEK Transport EKU specified.

11. Certificate Profiles

We need to put in text to express the following items:

DNS or IPAddr subject alt name to be present

Have one of four EKUs

Plasma Token EKU - Signals that it can sign and/or encrypt a plasma object

Plasma Secure Session - Use for the TLS session

Plasma CEK Transport - Used for transporting the CEK to the server in high security situations

MUST NOT have the anyPolicy EKU set

12. Message Transmission

Plasma messages are sent over a TCP connection using port TBD1 on the server. The client first setups up TLS on the connection, then sends the UTF8 encoded XML message over the TLS connection as an atomic message. The XML MUST be encoded as UTF8, however the Byte Order Mark (BOM) is sent. The response comes back on the same connection. The client is responsible for closing the TLS session and the TCP connection when either no more messages are to be sent to the server or a final indeterminate state has been reached.

If a Plasma server receives an XML request which is not well formed XML, the server if free to close the connection without first sending an error reply.

The Plasma server SHOULD support TLS resumption [RFC5077].

Plasma clients and server MUST support TLS 1.1 [RFC4346] and above. Implementations SHOULD NOT allow for the use of TLS 1.0 or SSL.

13. Plasma URI Scheme

13.1. Plasma URI Schema Syntax

The scheme name for is "plasma".

The syntax for the plasma URI Schema is:

URI = "plasma" ":" "//" authority path-empty

Using the ABNF defined in [RFC3986]. When the port component is absent, then the value of TBD1 will be used. The userinfo portion of the authority MUST be absent.

13.2. Definition of Operations

This schema is defined to provide the location of a Plasma server. The sole operation is to establish a connection to the Plasma server over which the protocol defined in this document is to run.

14. Security Considerations

To be supplied after we have a better idea of what the document looks like.

14.1. Plasma URI Schema Considerations

TBD

15. IANA Considerations

We define the following name spaces

New name space for the plasma documents urn:ietf:params:xml:ns:plasma

15.1. Plasma Action Values

A new registry is established for Plasma server action identifiers using the tag "actions". The full urn for the registry is "urn:ietf:params:xml:ns:plasma:actions". This registry operates under a specification required policy. All entries in this registry require the following elements:

- A string in camel case which identifies the action to be performed.
- An optional XML data structure used to carry the control data for the action.
- An optional XML data structure used to return the result of the action from the server.
- A document reference describing the steps to be taken by the server.

The registry will be initially populated with the following:

Action Id	Input Structure	Output Structure
GetRoleTokens	none	eps:RoleToken
GetSendCMSToken	eps:GetCMSToken	eps:CMSLockBox
ParseCMSToken	eps:CMSLockBox	eps:CMSKey
GetReplyToken	none	eps:RoleToken

When these actions are placed in an xacml:Request,

- the Category is "urn:oasis:names:tc:xacml:3.0:attribute-category:action",

- the AttributeId is "urn:ietf:params:xml:ns:plasma:actions",

- the DataType is "http://www.w3.org/2001/XMLSchema#string"

15.2. non

Define a new data name space urn:ietf:params:xml:ns:plasma:data

CMSToken

ChannelBinding

SMIME-Capabilities

Define a new name space for status codes at urn:ietf:params:xml:ns:plasma:status. The initial set of values is

authentication-error This identifier indicates that the authentication methods failed to successfully complete.

Define a new name space for obligations. The same namespace will be used both for obligations and for advice and the values may appear in either section.

signature-required This identifier indicates that that the encrypted body must contain a signature element. The data value of this type shall be "http://www.w3.org/2001/XMLSchema#hexBinary" and the data structure shall consist of a DER encoded CMSCapabilities structure [RFC5751] with the list of permitted signature algorithms. If there are no restrictions on the algorithms or the restriction is implicit, then the data value MAY be omitted.

encryption-algorithms see above

ambigous-identity The identity of the client is either not stated in a form the Plasma server understands, or there are multiple identities in the authentication data. To remedy this situation, the client includes an explicit identity in the xacml:Reqeust element.

We define a schema in appendix A at urn:ietf:params:xml:schema:plasma:1.0:RFCTBD

Define a new Status Code for use in the Status URI field.

urn:ietf:params:xml:ns:plasma:status:gss-api-response - This status is returned only with Indefinite responses. Indicates a

GSS-API response object was returned in the GSSAPIResponse token type. Will return until authentication has been completed.

15.3. Port Assignment

We request that IANA assign a new port for the use of this protocol.

Service name: plasma

Port Number: TBD1

Transport Protocol: TCP

Description: Plasma Service Protocol

Reference: This document

Assignee: iesg@ietf.org

Contact: chair@ietf.org

Notes: The protocol requires that TLS be used to communicate over this port. There is no provision for unsecure messages to be sent to this protocol.

16. Open Issues

List of Open Issues:

- JLS: Should we require that any SignatureProperty be present for XML Signature elements?
- JLS: Need to figure out an appropriate way to reference the assertion from a dig sig element. Could use a special version of RetrievalMethod with a transform, but that does not seem correct. May need to define a new KeyInfo structure to do it.
- JLS: Should X.509 certificates and attribute certificates be fully specified as an authentication method?
- JLS: Should a SignerInfo attribute be placed under the access-subject Category for a senders version and under Environment for a machine version? Currently both are under Data
- JLS: Need an obligation to say that CEK must be encrypted. Do we also need to have recipient info structures encrypted?

17. References

17.1. Normative References

[ABFAB] Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", Work In Progress draft-ietf-abfab-gss-eap-04, Oct 2011.

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

[I-D.schaad-plasma-cms] Schaad, J., "Email Policy Service ASN.1 Processing", Work In Progress draft-schaad-plamsa-cms, Jan 2011.

[XML-Signature] Roessler, T., Reagle, J., Hirsch, F., Eastlake, D., and D. Solo, "XML Signature Syntax and Processing (Second Edition)", World Wide Web Consortium Recommendation REC-xmldsig-core-20080610, June 2008, <http://www.w3.org/TR/2008/REC-xmldsig-core-20080610>.

[XML-C14N11] Boyer, J. and G. Marcy, "Canonical XML Version 1.1", World Wide Web Consortium Recommendation REC-xml-c14n11-20080502, May 2008, <http://www.w3.org/TR/2008/REC-xml-c14n11-20080502>.

[WS-TRUST] Lawrence, K., Kaler, C., Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and H. Granqvist, "WS-Trust 1.4", OASIS Standard ws-trust-200902, March 2007, <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html>.

[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>.

[I-D.freeman-plasma-requirements] Freeman, T., Schaad, J., and P. Patterson, "Requirements for Message Access Control", Work in progress draft-freeman-message-access-control, October 2011.

[OASIS-CORE]
Cantor, S., Ed., Kemp, J., Ed., Philpott, R., Ed., and E. Maler, Ed., "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.

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

[RFC7055] Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", RFC 7055, December 2013.

17.2. Informative References

[RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, May 2009.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4998] Gondrom, T., Brandner, R., and U. Pordesch, "Evidence Record Syntax (ERS)", RFC 4998, August 2007.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[SAML-XACML]
Anderson, A., Ed. and H. Lockhart, Ed., "SAML 2.0 profile of XACML v2.0", OASIS Standard access_control-xacml-2.0 -saml-profile-spec-os.pdf, February 2005.

[PlasmaBasicPolicy]
Anon, A., "IETF Defined Plasma Policies", February 2005.

[SOAP11] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., and D. Winer, "Simple Object Access Protocol (SOAP) 1.1", W3C NOTE NOTE-SOAP-20000508, May 2000.

[SOAP12] Lafon, Y., Gudgin, M., Hadley, M., Moreau, J., Mendelsohn, N., Karmarkar, A., and H. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", World Wide Web Consortium Recommendation REC-soap12-part1-20070427, April 2007, <http://www.w3.org/TR/2007/REC-soap12-part1-20070427>.

[I-D.ietf-emu-eap-tunnel-method] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel EAP Method (TEAP) Version 1", draft-ietf-emu-eap-tunnel-method-09 (work in progress), September 2013.

[RFC7029] Hartman, S., Wasserman, M., and D. Zhang, "Extensible Authentication Protocol (EAP) Mutual Cryptographic Binding", RFC 7029, October 2013.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[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>.

Appendix A. XML Schema

This appendix represents the entirety of the XML Schema for Plasma documents.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2007 rel. 3 sp1 (http://www.altova.com) by James
Schaad (exmsft) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:wst="http://
schemas.xmlsoap.org/ws/2005/02/trust" xmlns:eps="urn:ietf:params:ns:plasma:1.0"
xmlns:ds2="http://www.w3.org/2000/09/xmldsig#"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xenc="http://
www.w3.org/2001/04/xmlenc#" targetNamespace="urn:ietf:params:ns:plasma:1.0"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:documentation>
    The PlasmaRequest element is one of two top level elements defined by this
```

```
XSD schema.
    The PlasmaRequest element is sent from the client to the server in order to
  </xs:documentation>
  </xs:annotation>
  <xs:element name="PlasmaRequest" type="eps:RequestType"/>
  <xs:complexType name="RequestType">
    <xs:sequence>
```

```
      <xs:element ref="eps:Authentication" minOccurs="0"/>
      <xs:element ref="xacml:Request"/>
    </xs:sequence>
    <xs:attribute name="Version" type="xs:string" default="1.0"/>
  </xs:complexType>
  <xs:element name="PlasmaResponse" type="eps:ResponseType"/>
  <xs:complexType name="ResponseType">
    <xs:sequence>
      <xs:element ref="xacml:Response"/>
      <xs:element ref="eps:PlasmaReturnToken" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Version" type="xs:string" default="1.0"/>
  </xs:complexType>
  <xs:element name="PlasmaReturnToken" type="eps:PlasmaReturnTokenType"/>
  <xs:complexType name="PlasmaReturnTokenType">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax"/>
    </xs:sequence>
    <xs:attribute name="DecisionId" type="xs:string"/>
  </xs:complexType>
  <xs:element name="Authentication" type="eps:AuthenticationType"/>
  <xs:complexType name="AuthenticationType">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="saml:Assertion"/>
      <xs:element name="GSSAPI" type="xs:hexBinary"/>
      <xs:element name="RoleToken">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##any" processContents="lax"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element ref="ds2:Signature"/>
      <xs:element name="Other">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##other"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
  <xs:element name="RoleToken" type="eps:RoleTokenType"/>
  <xs:complexType name="RoleTokenType">
    <xs:sequence>
      <xs:element name="FriendlyName" type="xs:string"/>
      <xs:element name="PDP" type="xs:anyURI" maxOccurs="unbounded"/>
```

```
      <xs:choice>
```

```
        <xs:element name="PolicyList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Policy" type="eps:PolicyDescType"
maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element ref="eps:Policy"/>
        <xs:element ref="eps:PolicySet"/>
      </xs:choice>
      <xs:element ref="wst:RequestSecurityTokenResponse"/>
      <xs:element ref="xacml:Obligations" minOccurs="0"/>
      <xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="PolicyDescType">
    <xs:sequence>
      <xs:element name="FriendlyName" type="xs:string"/>
      <xs:element name="Options" minOccurs="0">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="xs:anyType">
              <xs:attribute name="optionsType" type="xs:anyURI" use="required"/
>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
  </xs:complexType>
  <xs:element name="PolicySet" type="eps:PolicySetType"/>
  <xs:complexType name="PolicySetType">
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="eps:Policy"/>
        <xs:element ref="eps:PolicySet"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>
  </xs:complexType>
  <xs:element name="Policy" type="eps:PolicyType"/>
  <xs:complexType name="PolicyType">
    <xs:sequence>
      <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
```

```
  </xs:complexType>
  <xs:element name="GetCMSToken" type="eps:CMSTokenRequestType"/>
```

```
  <xs:complexType name="CMSTokenRequestType">
    <xs:sequence>
      <xs:choice>
        <xs:element ref="eps:Policy"/>
        <xs:element ref="eps:PolicySet"/>
      </xs:choice>
      <xs:element name="Hash">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="ds2:DigestMethod"/>
            <xs:element ref="ds2:DigestValue" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="LockBox" type="eps:LockBoxType" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="CEK" type="xs:hexBinary" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="LockBox" type="eps:LockBoxType"/>
  <xs:complexType name="LockBoxType">
    <xs:sequence>
      <xs:element name="Subject" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:anySimpleType">
              <xs:attribute name="type" type="xs:string" use="required"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:choice>
        <xs:element name="CMSLockBox" type="xs:base64Binary"/>
        <xs:element name="XMLLockBox" type="xenc:EncryptedKeyType"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="CMSKey" type="eps:CMSKeyResponseType"/>
  <xs:complexType name="CMSKeyResponseType">
    <xs:sequence>
      <xs:element name="DisplayString" type="xs:string"/>
      <xs:choice>
        <xs:element name="CEK" type="xs:base64Binary"/>
        <xs:element name="CMSLockBox" type="xs:base64Binary"/>
                                <xs:element name="XMLLockBox"
type="enc:EncryptedKeyType"/>
                                <xs:any namespace="##other"
```

```
processContents="lax"/>
      </xs:choice>
      <xs:element ref="eps:RoleToken" minOccurs="0"/>
```

```
      <xs:element ref="xacml:Attributes" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="CMSToken" type="eps:CMSTokenResponseType"/>
  <xs:complexType name="CMSTokenResponseType">
    <xs:sequence>
      <xs:element name="CMSLockBox" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:base64Binary">
              <xs:attribute name="CMSType" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LockboxKey">
    <xs:sequence>
      <xs:choice>
        <xs:element name="X509Certificate" type="xs:base64Binary"/>
        <xs:element name="PGPKey" type="xs:base64Binary"/>
        <xs:element ref="ds2:KeyInfo"/>
      </xs:choice>
      <xs:element name="Capabilities" type="xs:base64Binary" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
        <xs:element name="XMLToken" type="eps:XMLTokenResponseType"/>
        <xs:complexType name="XMLTokenResponseType">
          <xs:sequence>
            <xs:element name="XMLLockBox" maxOccurs="unbounded"
type="xenc:EncryptedKeyType"/>
          </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Appendix B. Example: Get Roles Request

This section provides an example of a request message to obtain the set of roles for an individual named 'bart@simpsons.com'. The authentication provided in this is a SAML statement included in the SAML_Collection element.

```
<?xml version="1.0" encoding="UTF-8"?>
<PlasmaRequest xmlns="urn:ietf:schema:plasma:1.0"
    xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:schema:plasma:1.0 C:
\ietf\drafts\Schema\Plasma.xsd" >
  <Authentication>
      <WS-Token>123456</WS-Token>
    <!-- <saml:Assertion>....</saml:Assertion> -->
  </Authentication>
  <xacml:Request CombinedDecision="false" ReturnPolicyIdList="false">
    <xacml:Attributes Category="urn:oasis:names:tc:xaml:3.0:attribute-
catagory:action">
      <xacml:Attribute IncludeInResult="false" AttributeId="urn:plasma:action-
id">
        <xacml:AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">GetRoleTokens</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:environment">
      <xacml:Attribute AttributeId="urn:ietf:plasma:data:channel"
IncludeInResult="false">
        <xacml:AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#base64Binary">ABCDEFGH</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
  </xacml:Request>
</PlasmaRequest>
```

Appendix C. Example: Get Roles Response

This section provides an example response to a successful request for a role sets.

```
&#65279;<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eps:PlasmaResponse xmlns:eps="urn:ietf:params:ns:plasma:1.0">
  <xacml:Response xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <xacml:Result>
      <xacml:Decision>Permit</xacml:Decision>
    </xacml:Result>
  </xacml:Response>
  <eps:PlasmaReturnToken>
    <eps:RoleToken>
      <eps:FriendlyName>Role #1</eps:FriendlyName>
      <eps:PDP>plasma://localhost:8080</eps:PDP>
      <eps:PolicyList>
```

```
        <eps:Policy PolicyId="urn:example:PlasmaPolicies:Policy1">
          <eps:FriendlyName>Schaad Policy 1</eps:FriendlyName>
        </eps:Policy>
      </eps:PolicyList>
      <wst:RequestSecurityTokenResponse xmlns:wst="http://schemas.xmlsoap.org/
ws/2005/02/trust">
        <wst:RequestedSecurityToken>
          <ex:MyToken xmlns:ex="http://example.com/
SecurityToken">MCgMCzxDb250ZXh0IC8+AgEBMBYYFDEvMTAvMjAxMyA0OjIyOjAwIEFN</
ex:MyToken>
```

```
        </wst:RequestedSecurityToken>
        <wst:Lifetime>
          <wsu:Expires xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd">2013-01-10T04:22:00</wsu:Expires>
        </wst:Lifetime>
      </wst:RequestSecurityTokenResponse>
      <Obligations xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
        <Obligation ObligationId="Obligation-Fred" />
      </Obligations>
    </eps:RoleToken>
  </eps:PlasmaReturnToken>
  <eps:PlasmaReturnToken>
    <eps:RoleToken>
      <eps:FriendlyName>Plasma Basic Policy</eps:FriendlyName>
      <eps:PDP>plasma://localhost:8080</eps:PDP>
      <eps:PolicyList>
        <eps:Policy PolicyId="urn:ietf:ns:plasma:policy:basic">
          <eps:FriendlyName>Plasma Basic Policy</eps:FriendlyName>
        </eps:Policy>
        <eps:Policy PolicyId="urn:example:PlasmaPolicies:Policy1">
          <eps:FriendlyName>Schaad Policy 1</eps:FriendlyName>
        </eps:Policy>
      </eps:PolicyList>
      <wst:RequestSecurityTokenResponse xmlns:wst="http://schemas.xmlsoap.org/
ws/2005/02/trust">
        <wst:RequestedSecurityToken>
          <ex:MyToken xmlns:ex="http://example.com/
SecurityToken">MCgMCzxDb250ZXh0IC8+AgEBMBYYFDEvMTAvMjAxMyA0OjIyOjAwIEFN</
ex:MyToken>
        </wst:RequestedSecurityToken>
        <wst:Lifetime>
          <wsu:Expires xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd">2013-01-10T04:22:00</wsu:Expires>
        </wst:Lifetime>
      </wst:RequestSecurityTokenResponse>
    </eps:RoleToken>
  </eps:PlasmaReturnToken>
</eps:PlasmaResponse>
```

In this example a role is returned that has two different policies that can be used by that role. Along with the role token, a binary secret is returned that is to be used in proving that the same entity is returning to use the roles.

Appendix D. Example: Get CMS Token Request

This section contains an example of a request from a client to a server for a CMS message token to be issued. The authentication for

the request is provided by using a WS-Trust token previously issued as part of a role request/response dialog. The request contains the following elements:

- A complex rule set is requested where permission to is to be granted to anyone who meets either of the two policies given.

- A specific recipient info structure is provided for a subject who's name is 'lisa@simpsons.com'. The details of the recipient info structure are skipped but it would be any encoding of a RecipientInfo structure from CMS.

- A generic key encryption key is provided for any other subject who meets the policies specified.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eps:PlasmaRequest xmlns:eps="urn:ietf:params:ns:plasma:1.0">
  <eps:Authentication>
    <eps:RoleToken>
      <ex:MyToken xmlns:ex="http://example.com/
SecurityToken">MCgMCzxDb250ZXh0IC8+AgEBMBYYFDEvMTAvMjAxMyAxOjI3OjEyIEFN</
ex:MyToken>
    </eps:RoleToken>
  </eps:Authentication>
  <xacml:Request CombinedDecision="false" ReturnPolicyIdList="false"
id="XACMLRequest" xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:action">
      <xacml:Attribute AttributeId="urn:ietf:params:xml:ns:params:actions"
IncludeInResult="false">
        <xacml:AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">GetCMSToken</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:environment">
      <xacml:Attribute AttributeId="urn:ietf:params:xml:ns:plasma:data:channel"
IncludeInResult="false">
        <xacml:AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#base64Binary">tls-unique</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Attributes Category="urn:ietf:params:xml:ns:params:data">
      <xacml:Attribute
AttributeId="urn:ietf:params:xml:ns:params:data:CMSTokenRequest"
IncludeInResult="false">
        <xacml:AttributeValue DataType="urn:ietf:params:ns:plasma:
1.0#CMSTokenRequestType">
          <eps:GetCMSToken>
            <eps:PolicySet PolicyCombiningAlgId="urn:oasis:names:tc:xacml:
3.0:policy-combining-algorithm:permit-overrides">
              <eps:Policy PolicyId="urn:example:PlasmaPolicies:Policy1" />
```

```
            </eps:PolicySet>
            <eps:Hash>
              <ds2:DigestMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#sha256" xmlns:ds2="http://www.w3.org/2000/09/xmldsig#" />
              <ds2:DigestValue xmlns:ds2="http://www.w3.org/2000/09/
xmldsig#">AQIDBAUGBwgJCg==</ds2:DigestValue>
            </eps:Hash>
            <eps:CEK>0102030405060708090A</eps:CEK>
          </eps:GetCMSToken>
        </xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
  </xacml:Request>
</eps:PlasmaRequest>
```

Appendix E. Example: Get CMS Token Response

This section contains an example of a response from a server to a client for a CMS message token to be issued. The token is returned in the CMSToken element. This element would then be placed into the CMS message being created by the client.

```
&#65279;<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eps:PlasmaResponse xmlns:eps="urn:ietf:params:ns:plasma:1.0">
  <xacml:Response xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <xacml:Result>
      <xacml:Decision>Permit</xacml:Decision>
    </xacml:Result>
  </xacml:Response>
  <eps:PlasmaReturnToken>

<eps:CMSLockbox>MIIQJAYJKoZIhvcNAQcCoIIQFTCCEBECAQExCzAJBgUrDgMCGgUAMIIDOQYJKoZIhvcNAQcDoIIDKjCC
qFbvzz7xxan6Q01By/J8X12Mpq00jLVst0+mGl7cmsBknS6TXC13638r8ow904GMB/
1YzmWVYs4Pc+p9l7UJ0MFjhVULuahMbwrpEEFg90GBvZzZXKy8syxTcyh3TwCMTpYHOJxz9DfowvSJi2TPUiXG0mXzzMkbS3
cYJpVDlQqMCOVui7UMlQAz3LQLa9GINTzs1I5j8uqPDwPKxKmWNJ5AYj3jb6uLsf0tD1h+mCKotjdVsC0Jx05xZ53UCYPg3K
hu8psH7Njq3aZ6McxgeBFKxswSD3ffipEWkwLyN0heyhvIn3/
prEsAwggEnBgsqhkiG9w0BB4aNFzAUBggqhkiG9w0DBwQIirvrkunYtn+AggEAlPZGLqxBvE2sdmmzUfAljJpKredC3fUxXg
miOwNTCUNrXg82s1NYfhWAQEuTxDtuGq7Lwd70fohcX0mXgxGbqlaPjEVzhUQwZvJfn1r7oosJ5qzO59sKStEntQdYR5cyYX
Lt7xRyB1HMz+eeLTl6amF1lO8VOITkAEOeI9noaePDheHMS7k0xMQMEMHYU1TN/
09/2RSbMY740MEDNpidtomFv4gvhWWzGrzYNPFNtHQh/
4UDhqXl9eJ+MOXRdGupV9vdt6RhGKC6krszfMV9O0vHzh750XwqxtQ38FolZ6CCChEwggSKMIIDcqADAgECAhAn9OoR9HqGx
KYFfKTzMmkMpW7DE8wvZigC4vlbhuiRvp4vKJvq1lepS/Pytptqi/
rrKGzaqq3Lmc1i3nhHmmI4uZGzaCl6r4LznY6eg6b6vzaJ1s9cx8i5khhxkzzabGoLhu21DEgLLyCio6kDqXXiUP8FlqvHXH
rz4cLwpMVnjNbWVDreCqS6A3ezZcj9HtN0YqoYymiTHqGFfvVHZcv4TVcodNI0/
zC27vZiMBSMLOsCAwEAAaOB4TCB3jAfBgNVHSMEGDAWgBStvZh6NLQm9/
rEJlTvA73gJMtUGjAdBgNVHQ4EFgQUiYJnfcSdJnAAS7RQSHzePa4Ebn0wDgYDVR0PAQH/
BAQDAgEGMA8GA1UdEwEB/
wQFMAMBAf8wewYDVR0fBHQwcjA4oDagNIYyaHR0cDovL2NybC5jb21vZG9jYS5jb20vQWRkVHJ1c3RFeHRlcm5hbENBUm9vc
Phvr9t2ReI3Gj3d5FkEeKwc9ING83rPW4RyLeVGwboYESnzy0l5hzy6bQWdpG1oT61yFDaoubH9v89/8KRqlDVQj8+BbVWx3
kL3COJftaVAaz0MS8bY37czIs6ZuEJC3Wf5F6aAJQHw4/
TenM9btn6NwcLjv8Ts3+Ao7jqBMKpSZEZekQ8k1Sp67cPsprMlxBbP71XaDq/
9H6m4ZYbT2WR+X+LpUEwgDMjqHyuzCCBX8wggRnoAMCAQICEQDVeeyr0T13xrMgLQj+cJ0iMA0GCSqGSIb3DQEBBQUAMIGuM
WhKcrZJIcM9c/opX6Mi7V44v38hFxwurjJplF74xLp5IqhaPGXHf0t8Yas/
F06UnZyMlXQ197jCIkqChfqnOxDJ+7zASoveRt6aNyFJRy3eJyYasqbURvmdBYom1UOYxfWrIM9VaulvW5TCiJIpbKiEwdIc
OgtFXB7C+IjIw1C7ZTxBD9Q8g9RRr/
TvmhGQ1Ru1BL7g52+Hs1vrAtjENxwmJwrZSsJHgMw5FfDbVLrwOI97iCWnrQ8acFiS5iRFGqWzQCJyWhJFHYvuw7RzQg761+
FEfMA4GA1UdDwEB/
wQEAwIFoDAMBgNVHRMBAf8EAjAAMCAGA1UdJQQZMBcGCCsGAQUFBwMEBgsrBgEEAbIxAQMFAjARBglghkgBhvhCAQEEBAMCE
yacR4Lkj35uHZ6kzndhsdcsug06879gOsW1OTFMSRYvHhYkwknLL4PKISxozVmiDvVYzYXqE+Gj4jZaNzbF8suowQCq7dS82
y1Y65mMlGQW+FOr8sIxxFz6lsvTxEul0VXUxbfAZ0MkrRxJH0Mw3W2QUAQ3dRR81Ba/
g8GNhawC+jUxggKrMIICpwIBATCBxDCBrjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAlVUMRcwFQYDVQQHEw5TYWx0IExha2Ug
zlevtNMuRrfnRJA30ecoXgXr33rEsbj9xEgH+xaJwBotwLC/i7wKutjc+Z8sfUt9X+H/
jNFyMFZwXYF2fgw6xKtSlkYXgTu9GioK6rpftHSifOd+aRJHMKJTeWAkwamF9XBmaWhQusDVHZmmd9uUEPNkUSo4T6r2atZ8
```

```
q1jvgVl53xKCTP+xa1usZarYUo2u974JjADN8uGlI4gv1wH1scojgXVYp8HWe6Uh0fYFdFRmefHj5rEkiB4POVVQoq8Bsk4s
eps:CMSLockbox>
  </eps:PlasmaReturnToken>
</eps:PlasmaResponse>
```

Appendix F. Example: Get CMS Key Request

```
&#65279;<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eps:PlasmaRequest xmlns:eps="urn:ietf:params:ns:plasma:1.0">
  <eps:Authentication>
    <eps:RoleToken>
      <ex:MyToken xmlns:ex="http://example.com/
SecurityToken">MCgMCzxDb250ZXh0IC8+AgEBMBYYFDEvMTAvMjAxMyAxOjI3OjEyIEFN</
ex:MyToken>
    </eps:RoleToken>
  </eps:Authentication>
  <xacml:Request CombinedDecision="false" ReturnPolicyIdList="false"
xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:action">
      <xacml:Attribute AttributeId="urn:ietf:params:xml:ns:params:actions"
IncludeInResult="false">
        <xacml:AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">GetCMSKey</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:environment">
      <xacml:Attribute AttributeId="urn:ietf:params:xml:ns:plasma:data:channel"
IncludeInResult="false">
        <xacml:AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#base64Binary">tls-unique</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Attributes Category="urn:ietf:params:xml:ns:params:data">
      <xacml:Attribute
AttributeId="urn:ietf:params:xml:ns:params:data:CMSKeyRequest"
IncludeInResult="false">
        <xacml:AttributeValue DataType="urn:ietf:params:ns:plasma:
1.0#CMSLockbox">

<eps:CMSLockbox>MIIQJAYJKoZIhvcNAQcCoIIQFTCCEBECAQExCzAJBgUrDgMCGgUAMIIDOQYJKoZIhvcNAQcDoIIDKjCC
qFbvzz7xxan6Q01By/J8X12Mpq00jLVst0+mGl7cmsBknS6TXC13638r8ow904GMB/
1YzmWVYs4Pc+p9l7UJ0MFjhVULuahMbwrpEEFg90GBvZzZXKy8syxTcyh3TwCMTpYHOJxz9DfowvSJi2TPUiXG0mXzzMkbS3
cYJpVDlQqMCOVui7UMlQAz3LQLa9GINTzs1I5j8uqPDwPKxKmWNJ5AYj3jb6uLsf0tD1h+mCKotjdVsC0Jx05xZ53UCYPg3K
hu8psH7Njq3aZ6McxgeBFKxswSD3ffipEWkwLyN0heyhvIn3/
prEsAwggEnBgsqhkiG9w0BB4aNFzAUBggqhkiG9w0DBwQIirvrkunYtn+AggEAlPZGLqxBvE2sdmmzUfAljJpKredC3fUxXg
miOwNTCUNrXg82s1NYfhWAQEuTxDtuGq7Lwd70fohcX0mXgxGbqlaPjEVzhUQwZvJfn1r7oosJ5qzO59sKStEntQdYR5cyYX
Lt7xRyB1HMz+eeLTl6amF1lO8VOITkAEOeI9noaePDheHMS7k0xMQMEMHYU1TN/
09/2RSbMY740MEDNpidtomFv4gvhWWzGrzYNPFNtHQh/
4UDhqXl9eJ+MOXRdGupV9vdt6RhGKC6krszfMV9O0vHzh750XwqxtQ38FolZ6CCChEwggSKMIIDcqADAgECAhAn9OoR9HqGx
```

```
KYFfKTzMmkMpW7DE8wvZigC4vlbhuiRvp4vKJvq1lepS/Pytptqi/
rrKGzaqq3Lmc1i3nhHmmI4uZGzaCl6r4LznY6eg6b6vzaJ1s9cx8i5khhxkzzabGoLhu21DEgLLyCio6kDqXXiUP8FlqvHXH
rz4cLwpMVnjNbWVDreCqS6A3ezZcj9HtN0YqoYymiTHqGFfvVHZcv4TVcodNI0/
zC27vZiMBSMLOsCAwEAAaOB4TCB3jAfBgNVHSMEGDAWgBStvZh6NLQm9/
rEJlTvA73gJMtUGjAdBgNVHQ4EFgQUiYJnfcSdJnAAS7RQSHzePa4Ebn0wDgYDVR0PAQH/
BAQDAgEGMA8GA1UdEwEB/
wQFMAMBAf8wewYDVR0fBHQwcjA4oDagNIYyaHR0cDovL2NybC5jb21vZG9jYS5jb20vQWRkVHJ1c3RFeHRlcm5hbENBUm9vc
Phvr9t2ReI3Gj3d5FkEeKwc9ING83rPW4RyLeVGwboYESnzy0l5hzy6bQWdpG1oT61yFDaoubH9v89/8KRqlDVQj8+BbVWx3
kL3COJftaVAaz0MS8bY37czIs6ZuEJC3Wf5F6aAJQHw4/
TenM9btn6NwcLjv8Ts3+Ao7jqBMKpSZEZekQ8k1Sp67cPsprMlxBbP71XaDq/
9H6m4ZYbT2WR+X+LpUEwgDMjqHyuzCCBX8wggRnoAMCAQICEQDVeeyr0T13xrMgLQj+cJ0iMA0GCSqGSIb3DQEBBQUAMIGuM
WhKcrZJIcM9c/opX6Mi7V44v38hFxwurjJplF74xLp5IqhaPGXHf0t8Yas/
F06UnZyMlXQ197jCIkqChfqnOxDJ+7zASoveRt6aNyFJRy3eJyYasqbURvmdBYom1UOYxfWrIM9VaulvW5TCiJIpbKiEwdIc
OgtFXB7C+IjIw1C7ZTxBD9Q8g9RRr/
TvmhGQ1Ru1BL7g52+Hs1vrAtjENxwmJwrZSsJHgMw5FfDbVLrwOI97iCWnrQ8acFiS5iRFGqWzQCJyWhJFHYvuw7RzQg761+
FEfMA4GA1UdDwEB/
wQEAwIFoDAMBgNVHRMBAf8EAjAAMCAGA1UdJQQZMBcGCCsGAQUFBwMEBgsrBgEEAbIxAQMFAjARBglghkgBhvhCAQEEBAMCB
yacR4Lkj35uHZ6kzndhsdcsug06879gOsW1OTFMSRYvHhYkwknLL4PKISxozVmiDvVYzYXqE+Gj4jZaNzbF8suowQCq7dS82
y1Y65mMlGQW+FOr8sIxxFz6lsvTxEul0VXUxbfAZ0MkrRxJH0Mw3W2QUAQ3dRR81Ba/
g8GNhawC+jUxggKrMIICpwIBATCBxDCBrjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAlVUMRcwFQYDVQQHEw5TYWx0IExha2Ug
zlevtNMuRrfnRJA30ecoXgXr33rEsbj9xEgH+xaJwBotwLC/i7wKutjc+Z8sfUt9X+H/
jNFyMFZwXYF2fgw6xKtSlkYXgTu9GioK6rpftHSifOd+aRJHMKJTeWAkwamF9XBmaWhQusDVHZmmd9uUEPNkUSo4T6r2atZ8
q1jvgVl53xKCTP+xa1usZarYUo2u974JjADN8uGlI4gv1wH1scojgXVYp8HWe6Uh0fYFdFRmefHj5rEkiB4POVVQoq8Bsk4s
eps:CMSLockbox>
        </xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
  </xacml:Request>
</eps:PlasmaRequest>
```

Appendix G. Example: Get CMS KeyResponse

```
&#65279;<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eps:PlasmaResponse xmlns:eps="urn:ietf:params:ns:plasma:1.0">
  <xacml:Response xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <xacml:Result>
      <xacml:Decision>Permit</xacml:Decision>
    </xacml:Result>
  </xacml:Response>
  <eps:PlasmaReturnToken>
    <CMSKey:eps xmlns:CMSKey="urn:ietf:params:ns:plasma:1.0">
      <eps:DisplayString>Schaad Policy 1</eps:DisplayString>
      <eps:CEK>AQIDBAUGBwgJCg==</eps:CEK>
    </CMSKey:eps>
  </eps:PlasmaReturnToken>
</eps:PlasmaResponse>
```

Appendix H. Enabling the MultiRequests option

NOTE: RFC Editor please remove this section prior to publication. This section exists as a note to the author to make sure that it can be done. It will be published as a separate document if desired.

One of the issues in doing multiple requests in a single message is the issue of correlation between the request and the results. We have make this issue even worse by the fact that we are return results that are not input attributes for the decision and that we are not returning as attributes of the decision.

The best way to deal with this is by putting tags into the request and reflect them in the return values for the response. The only place that this does not work is for the GSS-API response token as this element would normally be part of the response of multiple requests. You want to finish that authentication step before issuing final decisions if the input is needed as part of that decision.

With this in mind what we do is the following:

- Define a new data attribute for plasma as plasma-request-id. The category for it is urn:ietf:params:xml:ns:plasma:data. The type will be a string.

- When the new attribute is used, then the return attribute flag MUST be set on the attribute.

- There MUST be one entity of the new attribute, with a unique value, for each of the requests in the MultiRequest element.

- Exactly one of the new attributes MUST be referenced in each request in the MultiRequest element.

- The server copies the value of the attribute into the *** attribute of the returned token.

We could probably relax the restrictions if we know that the token can only be returned by one request, however using the token to correlate the request and the decision is still probably desired so that those values can be correlated.

Author's Address

Jim Schaad
Soaring Hawk Consulting

Email: ietf@augustcellars.com