Plasma Service Trust Processing  
draft-schaad-plasma-service-03

Abstract

Write Me

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 February 25, 2013.

Copyright Notice

Copyright (c) 2012 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 ................................. 4
   1.1. XML Nomenclature and Name Spaces ........ 4
   1.2. Requirements Terminology ................. 5
2. Components ................................. 6
   2.1. XACML 3.0 ............................. 6
   2.2. SAML .................................. 6
   2.3. WS-Trust 1.4 ........................... 7
3. Model ..................................... 8
   3.1. Sender Processing ........................ 8
   3.2. Receiving Agent Processing .............. 9
4. Protocol Overview ......................... 11
5. Plasma Request ............................. 12
   5.1. Authentication Element .................... 13
   5.1.1. SAML Assertion ........................ 15
   5.1.2. WS Trust Tokens ........................ 16
   5.1.3. XML Signature Element ................. 17
   5.1.4. GSS-API Element ....................... 18
   5.2. xacml:Request Element ..................... 18
6. Plasma Response Element ................. 20
   6.1. xacml:Response Element ................... 20
7. Role Token and Policy Acquisition ....... 23
   7.1. Role Token Request ....................... 23
   7.2. Request Role Token Response .............. 24
   7.2.1. RoleToken XML element .................. 25
   7.2.2. Email Address List Options ............ 28
8. Sending An Email ........................... 29
   8.1. Send Message Request ..................... 29
   8.1.1. CMS Message Token Data Structure .... 30
   8.1.2. XML Label Structure ................... 32
   8.2. Send Message Response .................... 37
9. Decoding A Message ......................... 39
   9.1. Requesting Message Key ................... 39
   9.2. Requesting Message Key Response ......... 40
10. Plasma Attributes ......................... 42
   10.1. Data Attributes ......................... 42
   10.1.1. Channel Binding Data Attribute ......... 42
   10.1.2. CMS Signer Info Data Attribute ........ 43
   10.1.3. S/MIME Capabilities Data Attribute .... 43
   10.2. Obligations and Advice ................... 44
   10.2.1. Signature Required .................... 44
   10.2.2. Encryption Required ................... 44
12. Message Transmission ....................... 47
13. Security Considerations ..................... 48
14. IANA Considerations ....................... 49
   14.1. Plasma Action Values ..................... 49
14.2. non .......................................................... 50
14.3. Port Assignment ........................................... 51
15. Open Issues .................................................. 52
16. References ................................................... 53
   16.1. Normative References ................................... 53
   16.2. Informative References ................................ 54
Editorial Comments .............................................. 57
Appendix A. XML Schema ......................................... 57
Appendix B. Example: Get Roles Request ...................... 62
Appendix C. Example: Get Roles Response ..................... 63
Appendix D. Example: Get CMS Token Request ................. 64
Appendix E. Example: Get CMS Token Response ............... 66
Appendix F. Example: Get CMS Key Request .................. 67
Appendix G. Example: Get CMS KeyResponse ................... 68
Appendix H. Enabling the MultiRequests option ............. 69
Author’s Address ............................................... 70
# 1. Introduction

## 1.1. XML Nomenclature and Name Spaces

The following name spaces are used in this document:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
<th>Specification(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>eps</td>
<td><a href="http://ietf.org/2011/plasma/">http://ietf.org/2011/plasma/</a></td>
<td>This Specification</td>
</tr>
<tr>
<td>wst</td>
<td><a href="http://docs.oasis-open.org/ws-sx/ws-trust/">http://docs.oasis-open.org/ws-sx/ws-trust/</a></td>
<td>[WS-TRUST]</td>
</tr>
<tr>
<td></td>
<td>200512</td>
<td></td>
</tr>
<tr>
<td>wsu</td>
<td><a href="http://docs.oasis-open.org/wss/2004/01/oasis">http://docs.oasis-open.org/wss/2004/01/oasis</a></td>
<td>[WS-Security]</td>
</tr>
<tr>
<td></td>
<td>is-200401-wss-wssecurity-utility-1.0.xsd</td>
<td></td>
</tr>
<tr>
<td>wss</td>
<td><a href="http://docs.oasis-open.org/wss/2004/01/oasis">http://docs.oasis-open.org/wss/2004/01/oasis</a></td>
<td>[WS-Security]</td>
</tr>
<tr>
<td></td>
<td>is-200401-wss-wssecurity-secext-1.0.xsd</td>
<td></td>
</tr>
<tr>
<td>wss</td>
<td><a href="http://docs.oasis-open.org/wss/oasis-wss-w">http://docs.oasis-open.org/wss/oasis-wss-w</a></td>
<td>[WS-Security]</td>
</tr>
<tr>
<td></td>
<td>security-secext-1.1.xsd</td>
<td></td>
</tr>
<tr>
<td>xac</td>
<td><a href="http://docs.oasis-open.org/xacml/3.0/xacml">http://docs.oasis-open.org/xacml/3.0/xacml</a></td>
<td>[XACML]</td>
</tr>
<tr>
<td></td>
<td>-3.0-core-spec-cs-01-en.html</td>
<td></td>
</tr>
<tr>
<td>ds</td>
<td><a href="http://www.w3.org/2000/09/xmldsig#">http://www.w3.org/2000/09/xmldsig#</a></td>
<td>[XML-Signature]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xen</td>
<td><a href="http://www.w3.org/2001/04/xmlenc#">http://www.w3.org/2001/04/xmlenc#</a></td>
<td>[XML-Encrypt]</td>
</tr>
<tr>
<td>wsa</td>
<td><a href="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing</a></td>
<td>[WS-Addressing]</td>
</tr>
<tr>
<td>xs</td>
<td><a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a></td>
<td>[XML-Schema1][XML-Schema2]</td>
</tr>
</tbody>
</table>
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**: 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:[anchor8] 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.[anchor9]

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. [anchor11]

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 [EPS-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. Receiving 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 EnvelopedData 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, while the protocol allows for signature operations to occur on sections of the message structure, the secure transport is responsible for providing the confidentiality and integrity protection services over the entire message.

Multiple dialogs may be run over a single secure transport. 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. This is an indicator that the dialog has not yet completed.
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:

```xml
<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 mailing 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 its 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 its 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:
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 associated 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 SubjectConfirmation 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

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[anchor18]
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 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.[anchor19] 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.
[anchor20].

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

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.

5.2. 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:
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 were 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" abstract="true">
  <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 that will return one or more PlasmaReturnToken elements. These tokens represent the answer, for a success, of the request.

A Plasma Return Token is the base type from which return values are derived. The optional attribute DecisionId is defined for correlation of requests and results in the event that multiple requests are made. This document defines the following items that are derived from this type:

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

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:

```xml
<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 a new set of status values to be returned. Formal declaration may be found in Section 14.

- `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. [anchor23]
xacml:AssociatedAdvice 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. [anchor24][anchor25]
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 14.1. 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:

```xml
<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 and 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:PlasmaReturnToken xsi:type="RoleToken">
    <eps:Name>Role Display Name</eps:Name>
    <eps:PDP>PDP Url names</eps:PDP>
    <eps:PolicyList>
      <eps:Policy>
        Details of a policy
      </eps:Policy>
      ... More policies ...
    </eps:PolicyList>
  </eps:PlasmaReturnToken>
</eps:PlasmaResponse>

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:
The eps:RoleToken element contains the following items:

Name  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.

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.

Label  contains a single specific label. This element is returned as part of a read message token to allow for replies to be formulated by an entity that cannot generally originate a message using the policy.

wst:RequestSecurityTokenResponse  contains the actual token itself.

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

Name  contains a display string that represents the policy. This element is localized in response to the TBD attribute in the TBD field.

PolicyId  This attribute 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 in 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[anchor27] . Examples of items that could be used as security tokens are SAML statements, 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 credentials, 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

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:plasma:options:emailAddrs". The type associated with the structure is a string. The string contains a space separated set of email addresses.

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 request token exchange 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 SHOULD be included in the initial message. The authorization is supplied by the role token returned above, however authentication may be required as well. The authentication data is placed here.

- The xacml:Request element 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:
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:Label"/>
      <xs:element ref="eps:Leaf"/>
    </xs:choice>
    <xs:element name="Hash">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="ds2:DigestMethod"/>
          <xs:element ref="ds2:DigestValue"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Recipient" type="eps:RecipientInfoType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="CEK" type="xs:hexBinary" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="RecipientInfo" type="eps:RecipientInfoType"/>
<xs:complexType name="RecipientInfoType">
  <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:element name="LockBox" type="xs:base64Binary"/>
  </xs:sequence>
</xs:complexType>

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

Category = "urn:oasis:names:tc:xacml:3.0:attribute-category:data"
AttributeId = "urn:ietf:plasma:data-id"
DataType = "http://www.w3.org/2001/XMLSchema#anyType"

The elements of the structure are used as:

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

Recipients
This optional element contains a 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.

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 RecipientInfoType structure are:

Subject
This element contains a subject identifier. 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. This will normally be an RFC 822 name. Multiple subject names can be provided for a single lock box.

LockBox 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.

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:

```xml
<xs:element name="Label" type="eps:LabelType"/>
<xs:complexType name="LabelType">
  <xs:sequence>
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="eps:Label"/>
      <xs:element ref="eps:Leaf"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="CombiningRule" type="eps:CombiningRuleType" use="required"/>
</xs:complexType>
<xs:simpleType name="CombiningRuleType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="and"/>
    <xs:enumeration value="or"/>
    <xs:enumeration value="except"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="Leaf" type="eps:LeafType"/>
<xs:complexType name="LeafType">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

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

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

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

The content of the Leaf element can be any XML element. The content is to be a set of options for the policy. The schema applied to the content is based on the policy selected.

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

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

CombiningRule
This attribute specifies the operation to be used in combining the elements of the tree together. This attribute can contain either a string or a URI. If it is a string it MUST be one of the strings defined in this document. Examples of URIs that could be used can be found in Section XX of [XACML]. Servers MUST be able to evaluate the set of logical operations defined here, but other combining rules can be implemented as well. The set of string values defined in this document are:

and  All policies referenced by this element MUST evaluate to accept.

or  At least one policy referenced by this element MUST evaluate to accept.

except  When this combining rule is used, there MUST be exactly two policy elements. The rule evaluates to accept if and only if the first element evaluates to accept and the second does not evaluate to accept.

TODO: Correct the following text to make sense in the XML space rather than in the ASN.1 space

8.1.2.1. Label Extensibility

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

However the extensibility is not handled the same in all cases. Each of the four different cases is outlined below.
8.1.2.1.1. Sender Composing

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

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

8.1.2.1.2. Sender to Email Policy Service

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

8.1.2.1.3. Recipient to Email Policy Service

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

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

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

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

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

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

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

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

8.1.2.1.4. Recipient User Agent Display

Recipient user agents may want to display the policy to the user. This policy may be communicated from the Email Policy Service to the recipient using the OneLabel ASN.1 structure or using a different type. The label has been successfully (or unsuccessfully) validated so access has been granted (or denied) to the message. At this point we are only talking about a user interface issue. The recipient user agent should make some type of provision for indicating that an operation was placed in that location of the tree, but the agent is not aware of what the operation is.
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:Decisions is always included in the response. If the 'Permit' value is returned then the eps:CMSToken element MUST be present.
- The PlasmaReturnToken element with an 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:

```xml
<eps:PlasmaResponse>
  <xacml:Response>
    <xacml:Result>
      <xacml:Decision>Permit</xacml:Decision>
    </xacml:Result>
  </xacml:Response>
  <eps:PlasmaReturnToken xsi:"eps:CMSTokenResponseType">
    <eps:RecipientInfo>234e34d3</eps:RecipientInfo>
  </eps:PlasmaReturnToken>
</eps:PlasmaResponse>
```

The schema used for returning a CMS token is:

```xml
<x:s:element name="CMSToken" type="eps:CMSTokenResponseType"/>
<x:s:complexType name="CMSTokenResponseType">
  <x:s:complexContent>
    <x:s:extension base="eps:PlasmaReturnTokenType">
      <x:s:sequence>
        <x:s:element name="RecipientInfo" maxOccurs="unbounded">
          <x:s:complexType>
            <x:s:complexType>
              <x:s:complexContent>
                <x:s:attribute name="CMSType" type="x:s:string"/>
              </x:s:complexType>
            </x:s:complexType>
          </x:s:complexType>
        </x:s:element>
      </x:s:sequence>
    </x:s:extension>
  </x:s:complexType>
</x:s:element>
</x:s: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 [Plasma] for examples of when this would occur). Additionally the Plasma server
 dapat return a KEKRecipientInfo structure with the Plasma Other Key attribute. (For details see
[EPS-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:

RecipientInfo

This element contains the ASN.1 encoding for a CMS RecipientInfo structure to be placed in the final message.

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.
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 [EPS-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.[anchor33] 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>...
  <xacml:Request>
    <xacml:Attributes Category="...:action">
      <xacml:Attribute AttributeId="...:action-id">
        <xacml:AttributeValue>ParseCMSSToken</xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Attribute Category="...:data">
      <xacml:Attribute AttributeId="...:data:CMSToken">
        <xacml:AttributeValue>
          Base64 encoded CMS Token Value
        </xacml:AttributeValue>
      </xacml:Attribute>
    </xacml:Attributes>
    <xacml:Request>
  </xacml:Request>
</eps:PlasmaRequest>

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

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:

```xml
<xs:element name="CMSKey" type="eps:CMSKeyResponseType"/>
<xs:complexType name="CMSKeyResponseType">
  <xs:complexContent>
    <xs:extension base="eps:PlasmaReturnTokenType">
      <xs:sequence>
        <xs:element name="DisplayString" type="xs:string"/>
        <xs:choice>
          <xs:element name="CEK" type="xs:hexBinary"/>
          <xs:element ref="eps:RecipientInfo"/>
        </xs:choice>
        <xs:element name="ResponseRoleToken" type="eps:RoleToken" minOccurs="0"/>
        <xs:element name="EncryptionRequestor" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</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.

- **KEK** returns the base64 encoded key encryption key.

- **eps:RecipientInfo** returns the encrypted key in the form of a CMS RecipientInfo structure.
10. Plasma Attributes

In this document a number of different XAMCL 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 will be 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:oasis:names:tc:xacml:3.0:attribute-category:data".
- The AttributeId attribute is "urn:ietf:params:xml:plasma:data-id: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 XML channel binding structure defined in Section 5.1.3.
10.1.2. CMS Signer Info Data Attribute

In many cases a policy stays 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 setup for 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:oasis:names:tc:xacml:3.0:attribute-category:data".
- The AttributeId attribute is "urn:ietf:params:xml:plasma:data-id:CMSSignerInfo".
- The Issuer attribute is absent.
- The DataType attribute is either base64Binary or hexBinary.

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 [SMIME-MSG].

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

- The Category attribute is "urn:oasis:names:tc:xacml:3.0:attribute-category:data".
- The AttributeId attribute is "urn:ietf:params:xml:plasma:data-id:SMIME-Capabilities".
The Issuer attribute is absent.

The DataType attribute is either base64binary or hexBinary.

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

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. Encryption 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
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.
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 sets 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 is 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. Security Considerations

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

We define the following name spaces

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

14.1. Plasma Action Values

A new registry is established for Plasma server action identifiers using the tag "action-id". 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:

<table>
<thead>
<tr>
<th>Action Id</th>
<th>Input Structure</th>
<th>Output Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetRoleTokens</td>
<td>none</td>
<td>eps:RoleToken</td>
</tr>
<tr>
<td>GetSendCMSToken</td>
<td>eps:MessageTokenRequest</td>
<td>eps:CMSToken</td>
</tr>
<tr>
<td>ParseCMSToken</td>
<td>eps:CMSToken</td>
<td>eps:CMSKeyToken</td>
</tr>
<tr>
<td>GetReplyToken</td>
<td>none</td>
<td>eps:RoleToken</td>
</tr>
</tbody>
</table>

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"

14.2. non

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

    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 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 CMCapabilities structure [SMIME-MSG] 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

ambiguous-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:Request element.

We define a schema in appendix A at
urn:ietf:params:xml:schema:plasma-RFC TBD

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

    urn:ietf:???: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.
14.3. 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.
15. Open Issues

List of Open Issues:

- JLS: URL definitions - do we need a new schema or do we just overload https? For our URLs - do we require that they be passed through the internationalization step first? Probably should since the locale information is on the server and the client might not agree.

- 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?
16. References

16.1. Normative References


16.2.  Informative References


Editorial Comments

[anchor7]  Trevor: I don't see Plasma using this type

[anchor8]  Trevor: What about attribute match?

[anchor9]  Trevor: This sounds like we ignore the mapping on the wire. There is no reason to mandate the mapping occurs inside the PDP.

[anchor11]  Brian: 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.

[anchor14]  jimsch: Remove this?

[anchor16]  Trevor: What is the difference between WS-trust SAML assertions and the SAML assertions above?

[anchor18]  Trevor: Would we use bearer tokens to reintroduce the client? The main protection of bearer tokens is if they are fresh so you have had little time to stream one

[anchor19]  trevor: 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.

[anchor20]  JLS: I don't know of any way to say use the asymmetric key that you authenticated with originally - can this be done?

[anchor23]  Trevor: What about audit obligations

[anchor24]  Trevor: Should we ignore it if present?

[anchor25]  JLS: 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.


[anchor28]  Trevor: did not think you have a revocation mechanism in SAML

[anchor33]  jls: We may want to require that a reply token always be
returned instead of just returning it on demand.
This appendix represents the entirety of the XML Schema for Plasma documents.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2007 sp2 (http://www.altova.com) by James Schaad (exmsft) -->
    <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" abstract="true">
        <xs:attribute name="DecisionId" type="xs:string"/>
    </xs:complexType>
    <!--  <xs:element name="RequestRoles">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="eps:Authentication" minOccurs="0"/>
            <xs:element name="Identity" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    </xs:element> -->
    <xs:element name="RequestMessageData">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Label" type="eps:LabelType"/>
                <xs:element name="RoleToken"/>
                <xs:element name="Recipients"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```
</xs:complexType>
</xs:element>  <!--
<xs:element name="ReadMessage"/>-->
<xs:complexType name="AuthenticationType">
<xs:choice maxOccurs="unbounded">
<xs:element ref="saml:Assertion"/>
<xs:element name="GSSAPI" type="xs:hexBinary"/>
<xs:element name="WS-Token">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:hexBinary">
        <xs:attribute name="tokenType" type="xs:anyURI"/>
      </xs:extension>
    </xs:simpleContent>
  </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:complexType name="RoleToken">
<xs:complexContent>
  <xs:extension base="eps:PlasmaReturnTokenType">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="PDP" type="xs:anyURI" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element ref="wst:RequestSecurityTokenResponse"/>
<xs:element ref="xacml:Obligations" minOccurs="0"/>
<xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
</xs:sequence>
<xs:element name="Name" 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:attribute name="PolicyId" type="xs:anyURI" use="required"/>
</xs:complexType>

<xs:element name="Label" type="eps:LabelType"/>
<xs:complexType name="LabelType">
  <xs:sequence>
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="eps:Label"/>
      <xs:element ref="eps:Leaf"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="CombiningRule" type="eps:CombiningRuleType" use="required"/>
</xs:complexType>

<xs:simpleType name="CombiningRuleType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="and"/>
    <xs:enumeration value="or"/>
    <xs:enumeration value="except"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="Leaf" type="eps:LeafType"/>
<xs:complexType name="LeafType">
  <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:Label"/>
      <xs:element ref="eps:Leaf"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<xs:element name="Hash">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ds2:DigestMethod"/>
      <xs:element ref="ds2:DigestValue"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Recipient" type="eps:RecipientInfoType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="CEK" type="xs:hexBinary" minOccurs="0"/>

<xs:element name="RecipientInfo" type="eps:RecipientInfoType"/>
<xs:element name="Subject" maxOccurs="unbounded"/>
<xs:element name="LockBox" type="xs:base64Binary"/>

<xs:element name="CMSKey" type="eps:CMSKeyResponseType"/>
<xs:complexType name="CMSKeyResponseType">
  <xs:complexContent>
    <xs:extension base="eps:PlasmaReturnTokenType">
      <xs:sequence>
        <xs:element name="DisplayString" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="CMSToken" type="eps:CMSTokenResponseType"/>
<xs:complexType name="CMSTokenResponseType">
  <xs:complexContent>
    <xs:extension base="eps:PlasmaReturnTokenType">
      <xs:sequence>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:sequence>
  <xs:element name="RecipientInfo" 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:extension>
</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: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>
</Authentication>
<xacml:Request CombinedDecision="false" ReturnPolicyIdList="false">
  <xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <xacml:Attribute 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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PlasmaResponse xmlns="urn:ietf:schema:plasma:1.0"
    xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:schema:plasma:1.0 C:\ietf\drafts\Schema\Plasma.xsd">
    <xacml:Response>
        <xacml:Result>
            <xacml:Decision>Permit</xacml:Decision>
        </xacml:Result>
        <PlasmaReturnToken xsi:type="RoleToken">
            <Name>Example Role Name</Name>
            <PDP>https://pdp.example.com/companyPolicies</PDP>
            <PolicyList>
                <Policy PolicyId="urn:example:policies:confidential">
                    <Name>Company Confidential</Name>
                </Policy>
                <Policy PolicyId="urn:example:policies:plasma">
                    <Name>Plasma Project</Name>
                </Policy>
            </PolicyList>
            <wst:RequestSecurityTokenResponse>
                <wst:TokenType>urn:plasma:roleToken</wst:TokenType>
                <wst:RequestedSecurityToken>....</wst:RequestedSecurityToken>
                <wst:Entropy><wst:BinarySecret>12345678</wst:BinarySecret></wst:Entropy>
                <wst:Lifetime><wsu:Expires>2012-02-01T00:00:00</wsu:Expires></wst:Lifetime>
            </wst:RequestSecurityTokenResponse>
        </PlasmaReturnToken>
    </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 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"?>
<PlasmaRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:schema:plasma:1.0 C:\ietf\drafts\Schema\Plasma.xsd"
xmni="urn:ietf:schema:plasma:1.0"
xmni:ds2="http://www.w3.org/2000/09/xmldsig#"
xmni:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
<Authentication>
<WS-Token>123456</WS-Token>
</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">GetCMSToken</xacml:AttributeValue>
</xacml:Attribute>
</xacml:Attributes>
<xacml:Attributes Category="urn:ietf:plasma:attribute-category:data">
<xacml:Attribute AttributeId="urn:plasma:data-id" IncludeInResult="false">
<xacml:AttributeValue DataType="xml">
<GetCMSToken>
<Label CombiningRule="or">
<Leaf Label="urn:example:policies:confidential"/>
<Leaf Label="urn:example:policies:plasma"/>
</Label>
<Hash>
<ds2:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds2:DigestMethod>
<ds2:DigestValue>ABCDEFG0</ds2:DigestValue>
</Hash>
<Recipient>
<Subject type="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
lisa@simpsons.com</Subject>
</Recipient>
<LockBox>FF33eeddccaa002234</LockBox>
</GetCMSToken>
</xacml:AttributeValue>
</xacml:Attribute>
</xacml:Attributes>
</xacml:Request>
</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.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PlasmaResponse xmlns="urn:ietf:schema:plasma:1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:schema:plasma:1.0 C:\ietf\drafts\Schema\Plasma.xsd"
 xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
 <xacml:Response>
   <xacml:Result>
     <xacml:Decision>Permit</xacml:Decision>
   </xacml:Result>
 </xacml:Response>
 <PlasmaReturnToken xsi:type="CMSTokenResponseType">
   <RecipientInfo>3425342352343243</RecipientInfo>
 </PlasmaReturnToken>
</PlasmaResponse>
```
Appendix F. Example: Get CMS Key Request

<?xml version="1.0" encoding="UTF-8"?>
xsi:schemaLocation="urn:ietf:schema:plasma:1.0 C:\ietf\drafts\schema\Plasma.xsd"
xmni:xaml="urn:oasis:names:tc:SAML:2.0:assertion"
xaml:xaml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
<Authentication>
<!-- <saml:Assertion></saml:Assertion> -->
</Authentication>
</GSSAPI></GSSAPI>
</Authentication>
<xacml:Request CombinedDecision="false" ReturnPolicyIdList="false">
<xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
<xacml:Attribute AttributeId="urn:plasma:action-id" IncludeInResult="false">
<xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">ParseCMSToken</xacml:AttributeValue>
</xacml:Attribute>
<xacml:Attribute AttributeId="urn:plasma:action-id" IncludeInResult="false">
<xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">GetReplyToken</xacml:AttributeValue>
</xacml:Attribute>
</xacml:Attributes>
<xacml:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:data">
<xacml:Attribute AttributeId="urn:plasma:data-id:CMSToken" IncludeInResult="false">
<xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#base64Binary">AABBDDEEFF1122344</xacml:AttributeValue>
</xacml:Attribute>
</xacml:Attributes>
</xacml:Request>
</PlasmaRequest>
Appendix G. Example: Get CMS KeyResponse

<?xml version="1.0" encoding="UTF-8"?>
<PlasmaResponse xmlns="urn:ietf:schema:plasma:1.0"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmns:schemaLocation="urn:ietf:schema:plasma:1.0 C:\ietf\drafts\Schema\Plasma.xsd"
xmns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
xmns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
xmns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
<xacml:Response>
  <xacml:Result>
    <xacml:Decision>Permit</xacml:Decision>
  </xacml:Result>
</xacml:Response>
<PlasmaReturnToken xsi:type="CMSKeyResponseType">
  <DisplayString>Company Confidential</DisplayString>
  <CEK>3425342352343243</CEK>
  <ResponseRoleToken>
    <Name>Message Response Role</Name>
    <PDP>https://pdp.example.com/companyPolicies</PDP>
    <Label CombiningRule="or">
      <Leaf PolicyId="urn:example:policies:confidential"/>
      <Leaf PolicyId="urn:example:policies:plasma"/>
    </Label>
  </ResponseRoleToken>
  <wst:RequestSecurityTokenResponse>
    <wst:TokenType>urn:plasma:roleToken</wst:TokenType>
    <wst:RequestedSecurityToken>ABCDEF</wst:RequestedSecurityToken>
    <wst:Lifetime><wsu:Expires>2012-02-01T00:00:00</wsu:Expires></wst:Lifetime>
  </wst:RequestSecurityTokenResponse>
</PlasmaReturnToken>
</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 made this issue even worse by the fact that we are returning 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:...: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