Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2014

T. Hardjono, Ed.
MIT

March 6, 2014

User-Managed Access (UMA) Profile of OAuth 2.0
draft-hardjono-oauth-umacore-09

Abstract

User-Managed Access (UMA) is a profile of OAuth 2.0. UMA defines how resource owners can control protected-resource access by clients operated by arbitrary requesting parties, where the resources reside on any number of resource servers, and where a centralized authorization server governs access based on resource owner policy.Met at advisory in Feb 2014.

Status of This Memo

This Internet-Draft will expire on September 7, 2014.

Copyright Notice

Table of Contents

1. Introduction

User-Managed Access (UMA) is a profile of OAuth 2.0 [OAuth2]. UMA defines how resource owners can control protected-resource access by clients operated by arbitrary requesting parties, where the resources reside on any number of resource servers, and where a centralized authorization server governs access based on resource owner policy. Resource owners configure authorization servers with access policies that serve as implicit authorization grants. Thus, the UMA profile of OAuth can be considered to encompass an authorization grant flow.

UMA serves numerous use cases where a resource owner outsources authorization for access to their resources, potentially even without the run-time presence of the resource owner. A typical example is the following: a web user (an end-user resource owner) can authorize a web app (client) to gain one-time or ongoing access to a protected resource containing his home address stored at a "personal data store" service (resource server), by telling the resource server to respect access entitlements issued by his chosen cloud-based authorization service (authorization server). The requesting party operating the client might be the resource owner himself, using a web or native app run by an e-commerce company that needs to know where to ship a purchased item, or it might be his friend who is using an online address book service to collect contact information, or it might be a survey company that uses an autonomous web service to compile population demographics. A variety of scenarios and use cases can be found in [UMA-usecases] and [UMA-casestudies].

Practical control of access among loosely coupled parties requires more than just messaging protocols. This specification defines only the technical "contract" between UMA-conforming entities; its companion Binding Obligations specification [UMA-obligations] defines the expected behaviors of parties operating and using these entities. Parties operating entities that claim to be UMA-conforming MUST provide documentation affirmatively stating their acceptance of the binding obligations contractual framework defined in the Binding Obligations specification.

In enterprise settings, application access management sometimes involves letting back-office applications serve only as policy enforcement points (PEPs), depending entirely on access decisions coming from a central policy decision point (PDP) to govern the access they give to requesters. This separation eases auditing and allows policy administration to scale in several dimensions. UMA

makes use of a separation similar to this, letting the resource owner serve as a policy administrator crafting authorization strategies for resources under their control.

In order to increase interoperable communication among the authorization server, resource server, and client, UMA defines several purpose-built APIs related to the outsourcing of authorization, themselves protected by OAuth in embedded fashion.

The UMA protocol has three broad phases, as shown in Figure 1.

The Three Phases of the UMA Profile of OAuth

```
                                              +--------------+
                                              |   resource   |
             +---------manage (A)------------ |     owner    |
             |                                +--------------+
             |         Phase 1:                      |
             |         protect a                control (B)
             |         resource                      |
             v                                       v
      +------------+               +----------+--------------+
      |            |               |protection|              |
      |  resource  |               |   API    | authorization|
      |   server   |<-protect (C)--|  (needs  |    server    |
      |            |               |   PAT)   |              |
      +------------+               +----------+--------------+
      | protected  |                          | authorization|
      | resource   |                          |     API      |
      |(needs RPT) |                          |  (needs AAT) |
      +------------+                          +--------------+
             ^                                       |
             |         Phases 2 and 3:         authorize (D)
             |         get authorization,            |
             |         access a resource             v
             |                                +--------------+
             +---------access (E)-------------|    client    |
                                              +--------------+

                                              requesting party
```

Figure 1

The phases work as follows:

Protect a resource (Described in Section 2.) The resource owner, who manages online resources at the resource server ("A"), introduces it to the authorization server so that the latter can

begin controlling the resources' protection. To accomplish this protection, the authorization server presents a protection API ("C") to the resource server. This API is OAuth-protected and requires a protection API token (PAT) for access. Out of band, the resource owner configures the authorization server with policies associated with the registered resource sets ("B").

Get authorization (Described in Section 3.) The client approaches the resource server seeking access to an UMA-protected resource. In order to access it successfully, the client must first use the authorization server's authorization API ("D") to obtain a requesting party token (RPT) on behalf of its requesting party, and the requesting party must supply to the authorization server any identity claims needed in order for the server to associate sufficient authorization data with that RPT. The API is OAuth-protected and requires an authorization API token (AAT) for access.

Access a resource (Described along with Phase 2 in Section 3.) The client successfully presents an RPT that has sufficient authorization data associated with it to the resource server, gaining access to the desired resource ("E"). In this sense, this phase is the "happy path" within phase 2. The nature of the authorization data varies according to the RPT profile in use.

Implementers have the oportunity to develop profiles (see Section 6) that specify and restrict various UMA protocol, RPT, and identity claim options, according to deployment and usage conditions.

1.1. Notational Conventions

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].

Unless otherwise noted, all the protocol properties and values are case sensitive.

1.2. Terminology

UMA introduces the following new terms and enhancements of OAuth term definitions.

resource owner

An OAuth resource that is the "user" in User-Managed Access. This is typically an end-user (a natural person) but it can also be a corporation or other legal person.

requesting party
An end-user, or a corporation or other legal person, that uses a client to seek access to a protected resource. The requesting party may or may not be the same party as the resource owner.

client
An application making protected resource requests with the resource owner's authorization and on the requesting party's behalf.

claim
A statement of the value or values of one or more identity attributes of a requesting party. A requesting party may need to provide claims to an authorization server in order to satisfy policy and gain permission for access to a protected resource.

resource set A set of one or more protected resources. In authorization policy terminology, a resource set is the "object" being protected.

scope A bounded extent of access that is possible to perform on a resource set. In authorization policy terminology, a scope is one of the potentially many "verbs" that can logically apply to a resource set ("object"). UMA associates scopes with labeled resource sets.

authorization data Data associated with a requesting party token that enables some combination of the authorization server and resource server to determine the correct extent of access to allow to a client. Authorization data is a key part of the definition of an RPT profile.

permission A scope of access over a particular resource set at a particular resource server that is being requested by, or granted to, a requesting party. In authorization policy terminology, a permission is an entitlement that includes a "subject" (requesting party), "verbs" (one or more scopes of access), and an "object" (resource set). A permission is one example of authorization data that an authorization server may issue.

permission ticket A correlation handle that is conveyed from an authorization server to a resource server, from a resource server to a client, and ultimately from a client to an authorization server, to enable the authorization server to

assess the correct resource owner policies to apply to a request for an authorization grant.

1.3. Achieving Distributed Protection Through APIs and Tokens

UMA's authorization server, resource server, and client roles are designed to work in an interoperable fashion when each is operated by an entirely separate party (for example, different organizations). For this reason, UMA specifies communications channels that the authorization server MUST implement as HTTP-based APIs that MUST use TLS and OAuth protection, and that the resource server MUST implement as an HTTP-based interface. UMA's use of TLS transport-layer security is governed by Section 1.6 of [OAuth2], which discusses deployment and adoption characteristics of different TLS versions. Three different types of access tokens are issued and used for a variety of purposes as part of these inter-role interactions.

It is also REQUIRED, in turn, for resource servers and clients on the requesting side of UMA interactions to use these channels, unless a profile is being used that enables API extensibility. Profiles that enable such alternatives are described in Section 5.

1.3.1. Protection API

The authorization server MUST present an TLS- and OAuth-protected, HTTP-based protection API for use by resource servers. The authorization server thus has an OAuth token endpoint and user authorization endpoint, and has the option to issue an OAuth refresh token along with any access tokens issued for these APIs. The authorization server MUST declare all of its protection API endpoints in its configuration data (see Section 1.4).

The protection API consists of three endpoints:

- OAuth resource set registration endpoint as defined by [OAuth-resource-reg]

- Endpoint for registering client-requested permissions

- OAuth token introspection endpoint as defined by [OAuth-introspection] and Section 3.3.1

An entity seeking protection API access MUST have the scope "http://docs.kantarainitiative.org/uma/scopes/prot.json". (This URI resolves to a JSON-encoded scope description, as defined in [OAuth-resource-reg]. The description is non-normative for UMA purposes.) An access token with at least this scope is called a protection API token (PAT) and an entity with this scope is

definitionally a resource server. A single entity can serve in both resource server and client roles if it has the appropriate OAuth scopes. If a request to an endpoint fails due to an invalid, missing, or expired PAT, or requires higher privileges at this endpoint than provided by the PAT, the authorization server responds with an OAuth error.

The authorization server MUST support the OAuth bearer token profile for PAT issuance, and MAY support other OAuth token profiles (for example, the SAML bearer token grant type [OAuth-SAML]). It MUST declare all supported token profiles and grant types for PAT issuance in its configuration data.

A PAT binds a resource owner, a resource server the owner uses for resource management, and an authorization server the owner uses for protection of resources at this resource server. It is not specific to any client or requesting party. The issuance of a PAT represents the approval of the resource owner for this resource server to trust this authorization server for protecting its resources belonging to this resource owner.

1.3.2. Authorization API

The authorization server MUST present an TLS- and OAuth-protected, HTTP-based authorization API for use by clients. The authorization server thus has an OAuth token endpoint and user authorization endpoint, and has the option to issue an OAuth refresh token along with any access tokens issued for these APIs. The authorization server MUST declare all of its authorization API endpoints in its configuration data (see Section 1.4).

The authorization API consists of two endpoints:

- Endpoint for RPT issuance

- Endpoint for requesting authorization

An entity seeking authorization API access MUST have the scope "http://docs.kantarainitiative.org/uma/scopes/authz.json". (This URI resolves to a JSON-encoded scope description, as defined in [OAuth-resource-reg]. The description is non-normative for UMA purposes.) An access token with at least this scope is called an authorization API token (AAT) and an entity with this scope is definitionally a client. A single entity can serve in both resource server and client roles if it has the appropriate OAuth scopes. If a request to an endpoint fails due to an invalid, missing, or expired AAT, or requires higher privileges at this endpoint than provided by the AAT, the authorization server responds with an OAuth error.

The authorization server MUST support the OAuth bearer token profile for AAT issuance, and MAY support other OAuth token profiles (for example, the SAML bearer token grant type [OAuth-SAML]). It MUST declare all supported token profiles and grant types for AAT issuance in its configuration data.

An AAT binds a requesting party, a client being used by that party, and an authorization server that protects resources this client is seeking access to on this requesting party's behalf. It is not specific to any resource server or resource owner. The issuance of an AAT represents the approval of this requesting party for this client to engage with this authorization server to supply claims, ask for authorization, and perform any other tasks needed for obtaining authorization for access to resources at all resource servers that use this authorization server. The authorization server is able to manage future processes of authorization and claims-caching efficiently for this client/requesting party pair across all resource servers they try to access; however, these management processes are outside the scope of this specification.

1.3.3. Protected Resource Interface

The resource server MAY present to clients whatever HTTP-based APIs or endpoints it wishes. To protect any of its resources available in this fashion using UMA, it MUST require a requesting party token (RPT) with sufficient authorization data for access.

This specification defines one RPT profile, call "bearer" (see Section 3.3.2), which the authorization server MUST support. It MAY support additional RPT profiles, and MUST declare all supported RPT profiles in its configuration data (see Section 1.4).

An RPT binds a requesting party, the client being used by that party, the resource server at which protected resources of interest reside, and the authorization server that protects those resources. It is not specific to a single resource owner, though its internal components are likely to be bound to individual resource owners, depending on the RPT profile in use.

1.3.4. Time-to-Live Considerations

The authorization server has the opportunity to manage the validity periods of access tokens that it issues, their corresponding refresh tokens where applicable, the individual data components associated with RPTs where applicable, and even the client credentials that it issues. Different time-to-live strategies may be suitable for different resources and scopes of access, and the authorization server has the opportunity to give the resource owner control over

lifetimes of tokens and authorization data issued on their behalf through policy. These options are all outside the scope of this specification.

1.4. Authorization Server Configuration Data

The authorization server MUST provide configuration data in a JSON [RFC4627] document that resides in an /uma-configuration directory at at its hostmeta [hostmeta] location. The configuration data documents conformance options and endpoints supported by the authorization server. (At the appropriate time, this section will instead profile whatever self-describing metadata specification OAuth adopts, for example, [OAuth-linktypes] or [OAuth-meta].)

The configuration data has the following properties.

version
: REQUIRED. The version of the UMA core protocol to which this authorization server conforms. The value MUST be the string "1.0".

issuer
: REQUIRED. A URI indicating the party operating the authorization server.

pat_profiles_supported
: REQUIRED. OAuth access token profiles supported by this authorization server for PAT issuance. The property value is an array of string values, where each string value is either a reserved keyword defined in this specification or a URI identifying an access token profile defined elsewhere. The reserved keyword "bearer" as a value for this property stands for the OAuth bearer token profile [OAuth-bearer]. The authorization server is REQUIRED to support this profile, and to supply this string value explicitly. The authorization server MAY declare its support for additional access token profiles for PATs.

aat_profiles_supported
: REQUIRED. OAuth access token profiles supported by this authorization server for AAT issuance. The property value is an array of string values, where each string value is either a reserved keyword defined in this specification or a URI identifying an access token profile defined elsewhere. The reserved keyword "bearer" as a value for this property stands for the OAuth bearer token profile [OAuth-bearer]. The authorization server is REQUIRED to support this profile, and to supply this string value explicitly. The authorization

server MAY declare its support for additional access token profiles for AATs.

rpt_profiles_supported

REQUIRED. UMA RPT profiles supported by this authorization server for RPT issuance. The property value is an array of string values, where each string value is either a reserved keyword defined in this specification or a URI identifying an RPT profile defined elsewhere. The reserved keyword "bearer" as a value for this property stands for the UMA bearer RPT profile defined in [OAuth-bearer]. The authorization server is REQUIRED to support this profile, and to supply this string value explicitly. The authorization server MAY declare its support for additional RPT profiles.

pat_grant_types_supported

REQUIRED. OAuth grant types supported by this authorization server in issuing PATs. The property value is an array of string values. Each string value MUST be one of the grant_type values defined in [OAuth2], or alternatively a URI identifying a grant type defined elsewhere.

aat_grant_types_supported

REQUIRED. OAuth grant types supported by this authorization server in issuing AATs. The property value is an array of string values. Each string value MUST be one of the grant_type values defined in [OAuth2], or alternatively a URI identifying a grant type defined elsewhere.

claim_profiles_supported

OPTIONAL. Claim formats and associated sub-protocols for gathering claims from requesting parties, as supported by this authorization server. The property value is an array of string values, which each string value is either a reserved keyword defined in this specification or a URI identifying a claim profile defined elsewhere.

uma_profiles_supported

OPTIONAL. UMA profiles supported by this authorization server. The property value is an array of string values, which each string value is either a reserved keyword defined in this specification or a URI identifying an UMA profile defined elsewhere. The reserved keywords "prot-ext", "authz-ext", and "rsrc-ext" as values for this property stand for the extensibility profiles defined, respectively, in Section 5.

dynamic_client_endpoint

OPTIONAL. The endpoint to use for performing dynamic client registration. Usage of this endpoint is defined by [DynClientReg]. The presence of this property indicates authorization server support for the dynamic client registration feature and its absence indicates a lack of support.

token_endpoint
REQUIRED. The endpoint URI at which the resource server or client asks the authorization server for a PAT or AAT, respectively. A requested scope of "http://docs.kantarainitiative.org/uma/scopes/prot.json" results in a PAT. A requested scope of "http://docs.kantarainitiative.org/uma/scopes/authz.json" results in an AAT. Usage of this endpoint is defined by [OAuth2].

user_endpoint
REQUIRED. The endpoint URI at which the resource server gathers the consent of the end-user resource owner or the client gathers the consent of the end-user requesting party, if the "authorization_code" grant type is used. Usage of this endpoint is defined by [OAuth2].

introspection_endpoint
REQUIRED. The endpoint URI at which the resource server introspects an RPT presented to it by a client. Usage of this endpoint is defined by [OAuth-introspection] and Section 3.3.1. A valid PAT MUST accompany requests to this protected endpoint.

resource_set_registration_endpoint
REQUIRED. The endpoint URI at which the resource server registers resource sets to put them under authorization manager protection. Usage of this endpoint is defined by [OAuth-resource-reg] and Section 2. A valid PAT MUST accompany requests to this protected endpoint.

permission_registration_endpoint
REQUIRED. The endpoint URI at which the resource server registers a client-requested permission with the authorization server. Usage of this endpoint is defined by Section 3.2. A valid PAT MUST accompany requests to this protected endpoint.

rpt_endpoint
REQUIRED. The endpoint URI at which the client asks the authorization server for an RPT. Usage of this endpoint is defined by Section 3.4.1. A valid AAT MUST accompany requests to this protected endpoint.

authorization_request_endpoint
REQUIRED. The endpoint URI at which the client asks to have authorization data associated with its RPT. Usage of this endpoint is defined in Section 3.4.2. A valid AAT MUST accompany requests to this protected endpoint.

Example of authorization server configuration data that resides at https://example.com/.well-known/uma-configuration (note the use of https: for endpoints throughout):

```
{
"version":"1.0",
"issuer":"https://example.com",
"pat_profiles_supported":["bearer"],
"aat_profiles_supported":["bearer"],
"rpt_profiles_supported":["bearer"],
"pat_grant_types_supported":["authorization_code"],
"aat_grant_types_supported":["authorization_code"],
"claim_profiles_supported":["openid"],
"dynamic_client_endpoint":"https://as.example.com/dyn_client_reg_uri",
"token_endpoint":"https://as.example.com/token_uri",
"user_endpoint":"https://as.example.com/user_uri",
"resource_set_registration_endpoint":"https://as.example.com/rs/rsrc_uri",
"introspection_endpoint":"https://as.example.com/rs/status_uri",
"permission_registration_endpoint":"https://as.example.com/rs/perm_uri",
"rpt_endpoint":"https://as.example.com/client/rpt_uri",
"authorization_request_endpoint":"https://as.example.com/client/perm_uri"
}
```

Authorization server configuration data MAY contain extension properties that are not defined in this specification. Extension names that are unprotected from collisions are outside the scope of this specification.

2. Protecting a Resource

The resource owner, resource server, and authorization server perform the following actions to put resources under protection. This list assumes that the resource server has discovered the authorization server's configuration data and endpoints as needed.

1. The authorization server issues client credentials to the resource server. It is OPTIONAL for the client credentials to be provided dynamically through [DynClientReg]; alternatively, they MAY use a static process.

2. The resource server acquires a PAT from the authorization server. It is OPTIONAL for the resource owner to introduce the resource

server to the authorization server dynamically (for example, through a "NASCAR"-style user interface where the resource owner selects a chosen authorization server); alternatively, they MAY use a static process that may or may not directly involve the resource owner at introduction time.

3. In an ongoing fashion, the resource server registers any resource sets with the authorization server for which it intends to outsource protection, using the resource set registration endpoint of the protection API (see [OAuth-resource-reg]).

Note: The resource server is free to offer the option to protect any subset of the resource owner's resources using different authorization servers or other means entirely, or to protect some resources and not others. Additionally, the choice of protection regimes can be made explicitly by the resource owner or implicitly by the resource server. Any such partitioning by the resource server or owner is outside the scope of this specification.

Once a resource set has been placed under authorization server protection through the registration of a resource set description for it, and until such a description's deletion by the resource server, the resource server MUST limit access to corresponding resources, requiring sufficient authorization data associated with client-presented RPTs by the authorization server (see Section 3.1.2).

3. Getting Authorization and Accessing a Resource

An authorization server orchestrates and controls clients' access (on their requesting parties' behalf) to a resource owner's protected resources at a resource server, under conditions dictated by that resource owner.

The process of getting authorization and accessing a resource always begins with the client attempting access at a protected resource endpoint at the resource server. How the client came to learn about this endpoint is out of scope for this specification. The resource owner might, for example, have advertised its availability publicly on a blog or other website, listed it in a discovery service, or emailed a link to a particular intended requesting party.

The resource server responds to the client's access request with whatever its application-specific resource interface defines as a success response, either immediately or having first performed one or more embedded interactions with the authorization server. Depending on the nature of the resource server's response to an failed access attempt, the client and its requesting party engage in embedded

interactions with the authorization server before re-attempting access.

The interactions are as follows. Each interaction MAY be the last, if the client chooses not to continue pursuing the access attempt or the resource server chooses not to continue facilitating it.

- The client attempts to access a protected resource.
  - If the access attempt is unaccompanied by an RPT, the resource server responds immediately with an HTTP 401 (Unauthorized) response and instructions on where to go to obtain one.
  - If the access attempt was accompanied by an RPT, the resource server checks the RPT's status.
    - If the RPT is invalid, the resource server responds with an HTTP 401 (Unauthorized) response and instructions on where to go to obtain a token.
    - If the RPT is valid but has insufficient authorization data, the resource server registers a suitable requested permission on the client's behalf at the authorization server, and then responds to the client with an HTTP 403 (Forbidden) response and instructions on where to go to ask for authorization.
    - If the RPT is valid, and if the authorization data associated with the token is sufficient for allowing access, the resource server responds with an HTTP 2xx (Success) response and a representation of the resource.
- If the client (possessing no RPT or an invalid RPT) received a 401 response and an authorization server's location, after looking up its configuration data and endpoints as necessary, it requests an RPT from the RPT endpoint of the authorization API.
- If the client (posessing a valid RPT) received a 403 response and a permission ticket, it asks the authorization server for authorization data that matches the ticket using the authorization request endpoint of the authorization API. If the authorization server needs requesting party claims in order to assess this client's authorization, it engages in a claims-gathering flow.
  - If the client does not already have an AAT at the appropriate authorization server to be able to use its authorization API, it first obtains one.

The interactions are described in detail in the following sections.

3.1. Client Attempts to Access Protected Resource

This interaction assumes that the resource server has previously registered one or more resource sets that correspond to the resource to which access is being attempted.

The client attempts to access a protected resource (for example, when an end-user requesting party clicks on a thumbnail representation of the resource to retrieve a larger version). It is expected to discover, or be provisioned or configured with, knowledge of the protected resource and its location out of band. Further, the client is expected to acquire its own knowledge about the application-specific methods made available by the resource server for operating on this protected resource (such as viewing it with a GET method, or transforming it with some complex API call).

The access attempt either is or is not accompanied by an RPT.

3.1.1. Client Presents No RPT

Example of a request carrying no RPT:

```
GET /album/photo.jpg HTTP/1.1
Host: photoz.example.com
...
```

If the client does not present an RPT with the request, the resource server returns an HTTP 401 (Unauthorized) status code and providing the authorization server's URI in an "as_uri" property to facilitate authorization server configuration data discovery, including discovery of the endpoint where the client can request an RPT (Section 3.4.1).

For example:

```
HTTP/1.1 401 Unauthorized
   WWW-Authenticate: UMA realm="example",
    host_id="photoz.example.com",
    as_uri="https://as.example.com"
   ...
```

3.1.2. Client Presents RPT

Example of a request carrying an RPT using the UMA bearer RPT profile:

```
GET /album/photo.jpg HTTP/1.1
Authorization: Bearer vF9dft4qmT
Host: photoz.example.com
...
```

If the client presents an RPT with its request, the resource server MUST determine the RPT's status (see Section 3.3) before responding.

If the RPT is invalid, the resource server applies UMA protection by returning an HTTP 401 (Unauthorized) status code and providing the authorization server's URI in an "as_uri" property in the header, similarly to the case where no RPT was presented.

If the RPT is valid but has insufficient authorization data for the type of access sought, the resource server uses the protection API to register a requested permission with the authorization server that would suffice for that scope of access (see Section 3.2). It then responds with the HTTP 403 (Forbidden) status code and providing the authorization server's URI in an "as_uri" property in the header and the permission ticket it just received from the AM in the body in a JSON-encoded "ticket" property.

Example of the resource server's response after having registered a requested permission and received a ticket:

```
HTTP/1.1 403 Forbidden
WWW-Authenticate: UMA realm="example",
  host_id="photoz.example.com",
  as_uri="https://as.example.com"
  error="insufficient_scope"

{
"ticket": "016f84e8-f9b9-11e0-bd6f-0021cc6004de"
}
```

If the RPT's status is associated with authorization data that is sufficient for the access sought by the client, the resource server MUST give access to the desired resource.

Example of the resource server's response after having determined that the RPT is valid and associated with sufficient authorization data:

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
...
```

```
/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja
3kAAQAEAAAAPAAA/+4ADkFkb2JlAGTAAAAAAf
/bAIQABgQEBAUEBgUFBgkGBQYJCwgGBggLDAo
KCwoKDBAMDAwMDAwQDA4PEA8ODBMTFBQTExwb
```

The resource server MUST NOT give access where the token's status is not associated with sufficient authorization data for the attempted scope of access.

3.2. Resource Server Registers Requested Permission With Authorization Server

In response to receiving an access request accompanied by an RPT that has insufficient authorization data, the resource server uses the protection API's permission registration endpoint to register a permission with the authorization server that would be sufficient for the type of access sought. The authorization server returns a permission ticket for the resource server to give to the client in its response. The PAT provided in the API request implicitly identifies the resource owner ("subject") to which the permission applies.

The resource server uses the POST method at the endpoint. The body of the HTTP request message contains a JSON object providing the requested permission, using a format derived from the scope description format specified in [OAuth-resource-reg], as follows. The object has the following properties:

resource_set_id REQUIRED. The identifier for a resource set to which this client is seeking access. The identifier MUST correspond to a resource set that was previously registered.

scopes REQUIRED. An array referencing one or more identifiers of scopes to which access is needed for this resource set. Each scope identifier MUST correspond to a scope that was registered by this resource server for the referenced resource set.

Example of an HTTP request that registers a requested permission at the authorization server's permission registration endpoint:

```
   POST /host/scope_reg_uri/photoz.example.com HTTP/1.1
   Content-Type: application/json
   Host: as.example.com

   {
     "resource_set_id": "112210f47de98100",
     "scopes": [
         "http://photoz.example.com/dev/actions/view",
         "http://photoz.example.com/dev/actions/all"
     ]
   }
```

If the registration request is successful, the authorization server responds with an HTTP 201 (Created) status code and includes the Location header in its response as well as the "ticket" property in the JSON-formatted body.

The permission ticket is a short-lived opaque structure whose form is determined by the authorization server. The ticket value MUST be securely random (for example, not merely part of a predictable sequential series), to avoid denial-of-service attacks. Since the ticket is an opaque structure from the point of view of the client, the authorization server is free to include information regarding expiration time within the opaque ticket for its own consumption. When the client subsequently uses the authorization API to ask the authorization server for authorization data to be associated with its RPT, it will submit this ticket to the authorization server.

For example:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://as.example.com/permreg/host/photoz.example.com/
5454345rdsaa4543
...

{
"ticket": "016f84e8-f9b9-11e0-bd6f-0021cc6004de"
}
```

If the registration request is authenticated properly but fails due to other reasons, the authorization server responds with an HTTP 400 (Bad Request) status code and includes one of the following UMA error codes (see Section 4.2):

invalid_resource_set_id The provided resource set identifier was not found at the authorization server.

invalid_scope At least one of the scopes included in the request was not registered previously by this resource server.

3.3. Resource Server Determines RPT's Status

The resource server MUST determine a received RPT's status, including both its validity and, if valid, its associated authorization data, before giving or refusing access to the client. An RPT is associated with a set of authorization data that governs whether the client is authorized for access. The token's nature and format are dictated by its profile; the profile might allow it to be self-contained, such that the resource server is able to determine its status locally, or might require or allow the resource server to make a run-time introspection request of the authorization server that issued the token.

This specification makes one type of RPT REQUIRED for the authorization server to support: the UMA bearer token profile, as defined in Section 3.3.2. Implementers MAY define and use other RPT profiles.

3.3.1. Token Introspection

Within any RPT profile, when a resource server needs to introspect a token in a non-self-contained way to determine its status, it MAY require, allow, or prohibit use of the OAuth token introspection endpoint (defined by [OAuth-introspection]) that is part of the protection API, and MAY profile its usage. The resource server MUST use the POST method in interacting with the endpoint, not the GET method also defined by [OAuth-introspection].

3.3.2. RPT Profile: Bearer

This section defines the UMA bearer token profile. Following is a summary:

- Identifying URI: http://docs.kantarainitiative.org/uma/profiles/uma-token-bearer-1.0

- Profile author and contact information: Thomas Hardjono (hardjono@mit.edu)

- Updates or obsoletes: None; this profile is new.

- Keyword in HTTP Authorization header: "Bearer".

- Syntax and semantics of token data: As defined below. The token data format mainly involves time-bounded permissions.

- Token data association: The data associated to the on-the-wire token by reference and retrieved at run time by the resource server through profiled use of the OAuth token introspection endpoint [OAuth-introspection], as defined below.

- Token data processing: As defined in this section and throughout Section 3 of this specification.

- Grant type restrictions: None.

- Error states: As defined below.

- Security and privacy considerations: As defined in this section and throughout Section 3 of this specification.

- Binding obligations: Because this RPT profile is mandatory for authorization servers to implement, binding obligations related to the use of this token profile are documented in [UMA-obligations].

On receiving an RPT of the "Bearer" type in an authorization header from a client making an access attempt, the resource server introspects the token by using the token introspection endpoint of the protection API. The PAT used by the resource server to make the introspection request provides resource-owner context to the authorization server.

The authorization server responds with a JSON object with the structure dictated by [OAuth-introspection]. If the valid property has a "true" value, then the JSON object MUST also contain an extension property with the name "permissions" that contains an array of zero or more values, each of which is an object consisting of these properties:

resource_set_id REQUIRED. A string that uniquely identifies the resource set, access to which has been granted to this client on behalf of this requesting party. The identifier MUST correspond to a resource set that was previously registered as protected.

scopes REQUIRED. An array referencing one or more URIs of scopes to which access was granted for this resource set. Each scope MUST correspond to a scope that was registered by this resource server for the referenced resource set.

expires_at REQUIRED. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this permission will expire.

issued_at OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this permission was originally issued.

Example:

```
HTTP/1.1 200 OK
   Content-Type: application/json
   Cache-Control: no-store

   {
    "valid": true,
    "expires_at": "1256953732",
    "issued_at": "1256912345",
    "permissions": [
      {
        "resource_set_id": "112210f47de98100",
        "scopes": [
          "http://photoz.example.com/dev/actions/view",
          "http://photoz.example.com/dev/actions/all"
         ],
        "expires_at" : "1256923456"
      }
    ]
   }
```

3.4. Client Seeks Authorization for Access

In order to access a protected resource successfully, a client needs to present a valid RPT with sufficient authorization data for access. To get to this stage requires a number of previously successful steps:

1. The authorization server issues client credentials to the client. It is OPTIONAL for the client credentials to be provided dynamically through [DynClientReg]; alternatively, they MAY use a static process.

2. The client acquires an AAT.

3. The client uses the authorization API to acquire an RPT. See Section 3.4.1 for more detail.

4. The client uses the authorization API to ask for authorization, providing the permission ticket it got from the resource server. The authorization server associates authorization data with the client's RPT based on the permission ticket, the resource owner's operative policies, and the results of any claims-gathering flows. See Section 3.4.2 for more detail.

3.4.1. Client Obtains RPT

The client might need an RPT if it has never before requested an RPT for this combination of requesting party, resource server, and authorization server, or if it has lost control of a previously issued RPT and needs a refreshed one. It obtains an RPT by using the authorization API, performing a POST on the RPT endpoint and supplying its AAT in the header. No body is expected; if a body is present, the authorization server MAY ignore it.

Example of a request message containing an AAT:

```
POST /rpt HTTP/1.1
Host: as.example.com
Authorization: Bearer jwfLG53^sad$#f
...
```

The authorization server responds with an HTTP 201 (Created) status code and provides a new RPT.

For example:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "rpt": "sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv"
}
```

If the AAT provided in the header is the same as one provided for a previously issued still-valid RPT by this authorization server, the authorization server invalidates the old RPT and issues a new one.

On first issuance, the RPT is associated with no authorization data and thus does not convey any authorizations for access.

3.4.2. Client Asks for Authorization Data

Once in possession of an AAT for this authorization server, an RPT that applies to this requesting party for this resource server and this authorization server, and a permission ticket, the client uses

the authorization API to ask the authorization server to give it suitable authorization data for the sought-for access. It performs a POST on the authorization request endpoint, supplying its own AAT in the header and its RPT and the permission ticket in a JSON object with properties "rpt" and "ticket", respectively.

Example of a request message containing an AAT, an RPT, and a permission ticket:

```
POST /authz_request HTTP/1.1
Host: as.example.com
Authorization: Bearer jwfLG53^sad$#f
...

{
 "rpt": "sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv",
 "ticket": "016f84e8-f9b9-11e0-bd6f-0021cc6004de"
}
```

The authorization server uses the ticket to look up the details of the previously registered requested permission, maps the requested permission to operative resource owner policies based on the resource set identifier and scopes in it, undergoes any claims-gathering flows required (see Section 3.5), and ultimately responds to the request. The resource owner's policies at the authorization server amount to an implicit authorization grant in governing the issuance of authorization data. (The authorization server is also free to enable the resource owner to set policies that require the owner to provide a run-time authorization grant in the form of a consent interaction, mediated by the authorization server. This setting of policies and gathering of consent is outside the scope of this specification.)

The authorization server MUST base the addition of authorization data to RPTs on user policies. The nature of these policies is outside the scope of UMA, but generally speaking, they can be thought of as either independent of requesting-party features (for example, dictating access based on time of day or client identity) or dependent on requesting-party features (for example, dictating access based on whether they are over 18 or present a certain identifier). Such requesting-party features can potentially be collected in a claims-gathering flow.

Once the authorization server associates authorization data with the RPT, it responds with an HTTP 201 (Created) status code. If the authorization server chooses to invalidate the original RPT in response to the request and to issue a new one to be associated with the resulting authorization data, it MUST provide that refreshed RPT in the body.

Example when the authorization data has been added and the RPT has been refreshed:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "rpt": "sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv"
}
```

If the authorization server does not add the requested authorization data, it responds using the appropriate HTTP status code and UMA error code (see Section 4.2):

invalid_ticket The provided ticket was not found at the authorization server. The authorization server responds with the HTTP 400 (Bad Request) status code.

expired_ticket The provided ticket has expired. The authorization server responds with the HTTP 400 (Bad Request) status code.

not_authorized_permission The client is definitively not authorized for this authorization according to user policy. The authorization server responds with the HTTP 403 (Forbidden) status code.

need_claims The authorization server is unable to determine whether the client is authorized for this permission without gathering requesting party claims. The authorization server responds with the HTTP 403 (Forbidden) status code. The client is therefore not authorized, but has the opportunity to engage in a requesting party claims-gathering flow (see Section 3.5) to continue seeking authorization.

Example when the ticket has expired:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
...

{
  "status": "error",
  "error": "expired_ticket"
}
```

3.5. Claims-Gathering Flows

The authorization server has a variety of options for coming into possession of claims in order to satisfy the resource owner's policy; this specification does not dictate a single answer. For example, the authorization server could interact with the requesting party to gather claims, or could accept claims delivered by a client, or could perform a lookup in some external system. The process for requesting and providing claims is extensible and can have a variety of dependencies on the type of requesting party (for example, natural person or legal person) and the type of client (for example, browser, native app, or autonomously running web service).

This specification provides a required framework for extensibility through profiling. The authorization server MAY support any number of claim profiles, and SHOULD document the claim profiles it supports in its configuration data. For the business-level and legal implications of different claim profiles, see [UMA-obligations]. Optional claim profiles are defined in [UMAclaims].

A client is operated by an end-user in one of two typical situations:

- The requesting party is a natural person (for example, a friend or family member of the resource owner); the requesting party may even be the resource owner herself.

- The requesting party is a legal person such as a corporation, and the end-user operating the client is acting as an agent of that legal person (for example, a customer support specialist representing a credit card company).

Where a claim profile dictates end-user interaction, a further variety of options is possible. The end-user could be required to register for and/or log in to an account or personal profile, or fill in a questionnaire, or complete a purchase. Several of these operations could even be required, where the order is treated as significant for evaluating resource owner policies.

4. Error Messages

Ultimately the resource server is responsible for either granting the access the client attempted, or returning an error response to the client with a reason for the failure. [OAuth2] defines several error responses for a resource server to return. UMA makes use of these error responses, but requires the resource server to "outsource" the determination of some error conditions to the authorization server. This specification defines additional UMA-specific error responses that the authorization server may give to the resource server and

client as they interact with it, and that the resource server may give to the client.

4.1. OAuth Error Responses

When a resource server or client attempts to access one of the authorization server endpoints or a client attempts to access a protected resource at the resource server, it has to make an authenticated request by including an OAuth access token in the HTTP request as described in [OAuth2] Section 7.2.

If the request failed authentication, the authorization server or the resource server responds with an OAuth error message as described throughout Section 2 and Section 3.

4.2. UMA Error Responses

When a resource server or client attempts to access one of the authorization server endpoints or a client attempts to access a protected resource at the resource server, if the request is successfully authenticated by OAuth means, but is invalid for another reason, the authorization server or resource server responds with an UMA error response by adding the following properties to the entity body of the HTTP response:

error REQUIRED. A single error code. Values for this property are defined throughout this specification.

error_description OPTIONAL. Human-readable text providing additional information.

error_uri OPTIONAL. A URI identifying a human-readable web page with information about the error.

The following is a common error code that applies to several UMA-specified request messages:

invalid_request The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed. The authorization server MUST respond with the HTTP 400 (Bad Request) status code.

For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
...

{
  "error": "invalid_request",
  "error_description": "There is already a resource with this identifier.",
  "error_uri": "http://as.example.com/errors/resource_exists"
}
```

5. Profiles for API Extensibility

In some circumstances, it is desirable to couple UMA roles tightly. For example, an authorization server application might also need to act as a client application in order to retrieve protected resources so that it can present to resource owners a dashboard-like user interface that accurately guides the setting of policy; it might need to access itself-as-authorization server for that purpose. For another example, the same organization might operate both an authorization server and a resource server that communicate only with each other behind a firewall, and it might seek more efficient communication methods between them.

This section defines profiles that allow inter-role communications channels and methods to vary in these specific circumstances. This specification still REQUIRES authorization servers to issue PATs, AATs, and RPTs and associate authorization data with RPTs, and REQUIRES resource servers to give clients access only when RPTs are associated with sufficient authorization data. This is because, although tokens might not always appear on the wire in the normal fashion in these cases, they represent binding obligations that might involve additional parties unable to take part in these optimization opportunities (see [UMA-obligations]).

In circumstances where alternate communications channels are being used between independently implemented system entities, it is RECOMMENDED, for reasons of implementation interoperability, to define concrete extension profiles that build on these extensibility profiles (see Section 6.1).

An authorization server using any of the opportunities afforded by the protection and/or authorization API extensibility profile MUST declare use of each profile by supplying the relevant "uma_profiles_supported" values in its configuration data (see Section 1.4).

5.1. Protection API Extensibility Profile

This section defines a profile for UMA where the authorization server and resource server roles either reside in the same system entity or otherwise have a privileged communications channel between them. Following is a summary:

- Identifying URI: http://docs.kantarainitiative.org/uma/profiles/prot-ext-1.0

- Profile author and contact information: Mark Dobrinic (mdobrinic@cozmanova.com)

- Updates or obsoletes: None; this profile is new.

- Security considerations: If the entities do not use TLS but communicate across a transport layer as opposed to using internal same-entity communication, it is STRONGLY RECOMMENDED to use an alternate means of transport-layer security.

- Privacy considerations: If the relationship between the roles is established in a manner that does not involve the resource owner at all, they each may maliciously leverage this relationship to observe the resource owner's personally identifiable information held in each system.

- Error states: See below.

- Binding obligations: Any applicable binding obligations are documented in [UMA-obligations].

Using this profile, the resource server MAY use means other than the TLS- and OAuth-protected HTTP-based protection API to communicate with the authorization server. This involves the following opportunities:

- A PAT MAY be issued without requiring an OAuth flow to establish one.

- Resource sets MAY be registered (or configured) without requiring explicit use of the resource set registration endpoint or presentation of a PAT in any registration request.

- Registration of requested permissions MAY be accomplished without requiring explicit use of the permission registration endpoint or presentation of a PAT in any registration request.

- RPT introspection MAY be accomplished without requiring explicit use of the token introspection endpoint or presentation of a PAT in any introspection request.

- Error states MAY arise and be reported in a different fashion from any HTTP-, OAuth-, and UMA-defined errors related to the protection API.

An authorization server using any of the opportunities afforded by this profile MUST declare use of this profile by supplying the "prot-ext-1.0" value for one of its "uma_profiles_supported" values in its configuration data (see Section 1.4).

5.2. Authorization API Extensibility Profile

This section defines a profile for UMA where the authorization server and client roles either reside in the same system entity or otherwise have a privileged communications channel between them. Following is a summary:

- Identifying URI: http://docs.kantarainitiative.org/uma/profiles/authz-ext-1.0

- Profile author and contact information: Mark Dobrinic (mdobrinic@cozmanova.com)

- Updates or obsoletes: None; this profile is new.

- Security considerations: If the entities do not use TLS but communicate across a transport layer as opposed to using internal same-entity communication, it is STRONGLY RECOMMENDED to use an alternate means of transport-layer security.

- Privacy considerations: If the relationship between the roles is established in a manner that does not involve the requesting party at all, they each may maliciously leverage this relationship to observe the requesting party's personally identifiable information held in each system.

- Error states: See below.

- Binding obligations: Any applicable binding obligations are documented in [UMA-obligations].

Using this profile, the resource server MAY use means other than the TLS- and OAuth-protected HTTP-based authorization API to communicate with the authorization server. This involves the following opportunities:

- An AAT MAY be issued without requiring an OAuth flow to establish one.

- An RPT MAY be issued without requiring explicit use of the RPT endpoint or presentation of an AAT in any RPT request.

- Authorization data MAY be associated with the RPT without requiring explicit use of the authorization request endpoint or presentation of an AAT, RPT, or ticket in any request.

- The client MAY use alternate means of initating a claims-gathering flow with the authorization server. (Any further profiling of this profile might involve a claim profile as well; see Section 6.3.)

- Error states MAY arise and be reported in a different fashion from any HTTP-, OAuth-, and UMA-defined errors related to the authorization API.

An authorization server using any of the opportunities afforded by this profile MUST declare use of this profile by supplying the "authz-ext-1.0" value for one of its "uma_profiles_supported" values in its configuration data (see Section 1.4).

5.3. Resource Interface Extensibility Profile

This section defines a profile for UMA where the resource server and client roles either reside in the same system entity or otherwise have a privileged communications channel between them. Following is a summary:

- Identifying URI: http://docs.kantarainitiative.org/uma/profiles/rsrc-ext-1.0

- Profile author and contact information: Mark Dobrinic (mdobrinic@cozmanova.com)

- Updates or obsoletes: None; this profile is new.

- Security considerations: If the entities do not use TLS but communicate across a transport layer as opposed to using internal same-entity communication, it is STRONGLY RECOMMENDED to use an alternate means of transport-layer security.

- Privacy considerations: If the relationship between the roles is established in a manner that does not involve the authorization server at all, they each may maliciously leverage this

relationship to observe the resource owner's or requesting party's personally identifiable information held in each system.

- Error states: See below.

- Binding obligations: Any applicable binding obligations are documented in [UMA-obligations].

Using this profile, the resource server MAY use means other than an HTTP-based resource interface to communicate with the authorization server. This involves the following opportunities:

- Resource access attempts MAY be accomplished without requiring explicit use of the HTTP-based endpoint or presentation of an RPT.

- Error states MAY arise and be reported in a different fashion from any HTTP-, OAuth-, and UMA-defined errors related to the protected resource's interface.

An authorization server involved in deployments where resource servers and clients are known to be using opportunities afforded by the resource interface extensibility profile MAY declare use of this profile by supplying the "rsrc-ext-1.0" value for one of its "uma_profiles_supported" values in its configuration data (see Section 1.4).

6. Specifying Additional Profiles

This specification defines a protocol that has optional features. For implementation interoperability and to serve particular deployment scenarios, including sector-specific ones such as healthcare or e-government, third parties may want to define profiles of UMA that restrict these options.

Further, this specification creates extensibility points for RPT profiles and claim profiles, and third parties may likewise want to define their own. Different RPT profiles could be used, for example, to change the dividing line between authorization server and resource server responsibilities in controlling access. Different claim profiles could be used to customize sector-specific or population-specific (such as individual vs. employee) claim types that drive the types of policies resource owners could set.

It is not practical for this specification to standardize all desired profiles. However, to serve overall interoperability goals, the following sections provide guidelines for third parties that wish to specify UMA-related profiles.

6.1. Specifying Profiles of UMA

It is RECOMMENDED that profiles of UMA document the following information:

1. Specify a URI that uniquely identifies the profile.

2. Identify the responsible author and provide postal or electronic contact information.

3. Supply references to previously defined profiles that the profile updates or obsoletes.

4. Specify the set of interactions between endpoint entites involved in the profile, calling out any restrictions on ordinary UMA-conformant operations and any extension properties used in message formats.

5. Identify the legally responsible parties involved in each interaction and any new obligations imposed, in the fashion of [UMA-obligations].

6. Define any additional or changed error states.

7. Supply any additional security and privacy considerations, including analysis of threats and description of countermeasures.

8. Specify any conformance considerations.

See Section 5 for examples.

6.2. Specifying RPT Profiles

It is RECOMMENDED that RPT profiles document the following information:

1. Specify a URI that uniquely identifies the token profile.

2. Identify the responsible author and provide postal or electronic contact information.

3. Supply references to previously defined token profiles that the token profile updates or obsoletes.

4. Specify the keyword to be used in HTTP Authorization headers with tokens conforming to this profile.

5. Specify the syntax and semantics of the data that the authorization server associates with the token.

6. Specify how the token data is associated with, contained within, and/or retrieved by means of, the on-the-wire token string.

7. Specify processing rules for token data.

8. Identify any restrictions on grant types to be used with the token profile.

9. Define any additional or changed error states.

10. Supply any additional security and privacy considerations.

11. Specify any obligations specific to the token profile, in the fashion of [UMA-obligations].

12. Specify any conformance considerations.

See Section 3.3.2 for an example.

6.3. Specifying Claim Profiles

In addition to any requirements listed in Section 3.5, it is RECOMMENDED that claim profiles document the following information:

1. Specify a URI that uniquely identifies the claim profile.

2. Identify the responsible author and provide postal or electronic contact information.

3. Supply references to previously defined claim profiles that the claim profile updates or obsoletes.

4. Specify the syntax and semantics of claim data and requests for claim data.

5. Specify how an authorization server gathers the claims.

6. Define any additional or changed error states.

7. Supply any additional security and privacy considerations.

8. Specify any obligations specific to the claim profile, in the fashion of [UMA-obligations].

9. Specify any conformance considerations.

See [UMAclaims] for examples.

7. Security Considerations

This specification relies mainly on OAuth security mechanisms as well as transport-level encryption for protecting the protection and authorization API endpoints. Most PATs and AATs are likely to use OAuth bearer tokens. See [OAuth-threat] for more information.

This specification defines a number of JSON-based data formats. As a subset of the JavaScript scripting language, JSON data SHOULD be consumed through a process that does not dynamically execute it as code, to avoid malicious code execution. One way to achieve this is to use a JavaScript interpreter rather than the built-in JavaScript eval() function.

The issue of impersonation is a crucial aspect in UMA, particularly when entities are wielding bearer tokens that preclude proof-of-possession (of a secret or a cryptographic key). As such, one way to mitigate this risk is for the resource owner to require stronger claims to accompany any access request. For example, consider the case where Alice sets policies at the authorization server governing access to her resources by Bob. When Bob first seeks access and must obtain an RPT (for which the default RPT profile specifies a bearer token), Alice could set policies demanding that Bob prove his identity by providing a set of strong claims issued by a trusted attribute provider in order to get authorization data associated with that token.

Another issue concerns the use of the [OAuth2] implicit flow. In this case, Bob will have exposure to the token, and may maliciously pass the token to an unauthorized party. To mitigate this weakness and others, we recommend considering the following steps:

- Require that the Requesting Party (as defined in [UMA-obligations], meaning this party is able to take on legal obligations) legitimately represent the wielder of the bearer token. This solution is based on a legal or contractual approach, and therefore does not reduce the risk from the technical perspective.

- The authorization server, possibly with input from the resource owner, can implement tighter time-to-live strategies around the authorization data in RPTs. This is a classic approach with bearer tokens that helps to limit a malicious party's ability to intercept and use the bearer token. In the same vein, the authorization server could require claims to have a reasonable degree of freshness (which would require a custom claims profile).

- The strongest strategy is to disallow bearer-type RPTs within the UMA profile being deployed, by providing or requiring an RPT profile that requires use of a holder-of-key approach. In this way, the wielder of the token must engage in a live session for proof-of-possession.

For information about the additional technical, operational, and legal elements of trust establishment between UMA entities and parties, which affects security considerations, see [UMA-obligations].

8. Privacy Considerations

The authorization server comes to be in possession of resource set information (such as names and icons) that may reveal information about the resource owner, which the authorization server's trust relationship with the resource server is assumed to accommodate. However, the client is a less-trusted party -- in fact, entirely untrustworthy until authorization data is associated with its RPT. This specification depends on [OAuth-resource-reg], which recommends obscuring resource set identifiers in order to avoid leaking personally identifiable information to clients through the scope mechanism.

For information about the technical, operational, and legal elements of trust establishment between UMA entities and parties, which affects privacy considerations, see [UMA-obligations].

Additional considerations related to Privacy by Design concepts are discussed in [UMA-PbD].

9. Conformance

This section outlines conformance requirements for various entities implementing UMA endpoints.

This specification has dependencies on other specifications, as referenced under the normative references listed in this specification. Its dependencies on some specifications, such as OpenID Connect ([OIDCCore]), are optional depending on whether the feature in question is used in the implementation.

The authorization server's configuration data provides a machine-readable method for it to indicate certain of the conformance options it supports. Several of the configuration data properties allow for indicating extension features. Where this specification does not already require optional features to be documented, it is RECOMMENDED that authorization server developers and deployers document any

profiled or extended features explicitly and use configuration data to indicate their usage. See Section 1.4 for information about providing and extending the configuration data.

10. IANA Considerations

This document makes no request of IANA.

11. Acknowledgments

The current editor of this specification is Thomas Hardjono of MIT. The following people are co-authors:

- Paul C. Bryan, ForgeRock US, Inc. (former editor)
- Domenico Catalano, Oracle Corp.
- Mark Dobrinic, Cozmanova
- George Fletcher, AOL
- Maciej Machulak, Newcastle University
- Eve Maler, XMLgrrl.com
- Lukasz Moren, Newcastle University
- Christian Scholz, COMlounge GmbH (former editor)
- Mike Schwartz, Gluu
- Jacek Szpot, Newcastle University

Additional contributors to this specification include the Kantara UMA Work Group participants, a list of whom can be found at [UMAnitarians].

12. Issues

Issues are captured at the project's GitHub site ([1]).

13. References

13.1. Normative References

[DynClientReg]
Richer, J., "OAuth 2.0 Core Dynamic Client Registration", August 2013, <http://tools.ietf.org/html/draft-richer-oauth-dyn-reg-core>.

[OAuth-bearer]
"The OAuth 2.0 Authorization Framework: Bearer Token Usage", October 2012, <http://tools.ietf.org/html/rfc6750>.

[OAuth-introspection]
Richer, J., "OAuth Token Introspection", May 2013, <http://tools.ietf.org/html/draft-richer-oauth-introspection>.

[OAuth-resource-reg]
Hardjono, T., "OAuth 2.0 Resource Set Registration", December 2012, <https://tools.ietf.org/html/draft-hardjono-oauth-resource-reg>.

[OAuth-threat]
Lodderstedt, T., "OAuth 2.0 Threat Model and Security Considerations", January 2013, <http://tools.ietf.org/html/rfc6819>.

[OAuth2] Hardt, D., "The OAuth 2.0 Authorization Framework", October 2012, <http://tools.ietf.org/html/rfc6749>.

[OIDCCore]
Sakimura, N., "OpenID Connect Core 1.0", December 2013, <http://openid.net/specs/openid-connect-core-1_0.html>.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[UMA-obligations]
Maler, E., "Binding Obligations on UMA Participants", January 2013, <http://kantarainitiative.org/confluence/display/uma/UMA+Trust+Model>.

[UMAclaims]
Catalano, D., "Claim Profiles for User-Managed Access (UMA)", February 2014, <http://docs.kantarainitiative.org/uma/draft-uma-claim-profiles.html>.

[hostmeta]
Hammer-Lahav, E., "Web Host Metadata", October 2011, <http://tools.ietf.org/html/rfc6415>.

13.2. Informative References

[OAuth-SAML]
Campbell, B., "SAML 2.0 Bearer Assertion Profiles for OAuth 2.0", December 2013, <http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer>.

[OAuth-linktypes]
Mills, W., "Link Type Registrations for OAuth 2", February 2013, <http://tools.ietf.org/html/draft-wmills-oauth-lrdd>.

[OAuth-meta]
Sakimura, N., "JSON Metadata for OAuth Responses", February 2013, <http://tools.ietf.org/html/draft-sakimura-oauth-meta>.

[UMA-PbD] Maler, B., "Privacy by Design Implications of UMA", December 2013, <http://kantarainitiative.org/confluence/display/uma/Privacy+by+Design+Implications+of+UMA>.

[UMA-casestudies]
Maler, E., "UMA Case Studies", March 2013, <http://kantarainitiative.org/confluence/display/uma/Case+Studies>.

[UMA-usecases]
Maler, E., "UMA Scenarios and Use Cases", October 2010, <http://kantarainitiative.org/confluence/display/uma/UMA+Scenarios+and+Use+Cases>.

[UMAnitarians]
Maler, E., "UMA Participant Roster", April 2013, <http://kantarainitiative.org/confluence/display/uma/Participant+Roster>.

13.3. URIs

[1] https://github.com/xmlgrrl/UMA-Specifications/issues

[2] http://kantarainitiative.org/confluence/display/uma/UMA+1.0+Core+Protocol

Appendix A. Document History

NOTE: To be removed by RFC editor before publication as an RFC.

See [2] for a list of code-breaking and other major changes made to this specification at various revision points.

Author's Address

Thomas Hardjono (editor)
MIT

Email: hardjono@mit.edu