Federated Authorization for User-Managed Access (UMA) 2.0

Version:	2.0
Date:	2018-1-7
Editor:	Eve Maler, ForgeRock
Authors:	Maciej Machulak, HSBC Justin Richer, Bespoke Engineering

Abstract

This specification defines a means for an UMA-enabled authorization server and resource server to be loosely coupled, or federated, in a secure and authorized resource owner context.

Status of This Document

This technical specification is a Recommendation produced by the User-Managed Access Work Group and approved by the Membership of the Kantara Initiative according to its Operating Procedures.

Copyright Notice

Table of Contents

1. Introduction

This specification extends and complements [UMAGrant] to loosely couple, or federate, its authorization process. This enables multiple resource servers operating in different domains to communicate with a single authorization server operating in yet another domain that acts on behalf of a resource owner. A service ecosystem can thus automate resource protection, and the resource owner can monitor and control authorization grant rules through the authorization server over time. Further, authorization grants can increase and decrease at the level of individual resources and scopes.

Building on the example provided in the introduction in [UMAGrant], bank customer (resource owner) Alice has a bank account service (resource server), a cloud file system (different resource server hosted elsewhere), and a dedicated sharing management service (authorization server) hosted by the bank. She can manage access to her various protected resources by spouse Bob, accounting professional Charline, financial information aggregation company DecideAccount, and neighbor Erik (requesting parties), all using different client applications. Her bank accounts and her various files and folders are protected resources, and she can use the same sharing management service to monitor and control different scopes of access to them by these different parties, such as viewing, editing, or printing files and viewing account data or accessing payment functions.

This specification, together with [UMAGrant], constitutes UMA 2.0. This specification is OPTIONAL to use with the UMA grant.

1.1 Notational Conventions

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

Unless otherwise noted, all parameter names and values are case sensitive. JSON [RFC7159] data structures defined in this specification MAY contain extension parameters that are not defined in this specification. Any entity receiving or retrieving a JSON data structure SHOULD ignore extension parameters it is unable to understand. Extension names that are unprotected from collisions are outside the scope of this specification.

1.2 Abstract Flow

The UMA grant defined in [UMAGrant] enhances the abstract protocol flow of OAuth. This specification enhances the UMA grant by defining formal communications between the UMA-enabled authorization server and resource server as they act on behalf of the resource owner, responding to authorization and resource requests, respectively, by a client that is acting on behalf of a requesting party.

A summary of UMA 2.0 communications, combining the UMA grant with federated authorization, is shown in Figure 1.

```
                                            +------------------+
                                            |     resource     |
       +-----------manage (out of scope)----|       owner      |
       |                                    +------------------+
       |                                             |
       |                                             |
       |               protection                    |
       |               API access                 control
       |               token (PAT)             (out of scope)
       |                                             |
       v                                             v
+------------+                   +----------+------------------+
|            |                   |protection|                  |
|  resource  |                   |   API    |  authorization   |
|   server   |<----protect-------| (needs   |      server      |
|            |                   |   PAT)   |                  |
+------------+                   +----------+------------------+
| protected  |                              |       UMA        |
| resource   |                              |      grant       |
|(needs RPT) |         requesting           | (PCT optional)   |
+------------+         party token          +------------------+
       ^                 (RPT)              ^  persisted   ^
       |                                    |   claims     |
       |                                  push   token     |
       |                                 claim   (PCT)     |
       |                                 tokens         interact
       |                                    +--------+    for
       +-----------access--------------------| client |  claims
                                             +--------+ gathering
                                               +---------------+
                                               |  requesting   |
                                               |     party     |
                                               +---------------+
```

Figure 1: Federated Authorization Enhancements to UMA Grant Flow

This specification uses all of the terms and concepts in [UMAGrant]. This figure introduces the following new concepts:

protection API The API presented by the authorization server to the resource server, defined in this specification. This API is OAuth-protected.

protection API access token (PAT) An [RFC6749] access token with the scope `uma_protection`, used by the resource server as a client of the authorization server's protection API. The resource owner involved in the UMA grant is the same entity taking on the role of the resource owner authorizing issuance of the PAT.

1.3 HTTP Usage, API Security, and Identity Context

This specification is designed for use with HTTP [RFC2616], and for interoperability and security in the context of loosely coupled services and applications operated by independent parties in independent domains. The use of UMA over any protocol other than HTTP is undefined. In such circumstances, it is RECOMMENDED to define profiles or extensions to achieve interoperability among independent implementations (see Section 4 of [UMAGrant]).

The authorization server MUST use TLS protection over its protection API endpoints, as governed by [BCP195], which discusses deployment and adoption characteristics of different TLS versions.

The authorization server MUST use OAuth and require a valid PAT to secure its protection API endpoints. The authorization server and the resource server (as an OAuth client) MUST support bearer usage of the PAT, as defined in [RFC6750]. All examples in this specification show the use of bearer-style PATs in this format.

As defined in [UMAGrant], the resource owner -- the entity here authorizing PAT issuance -- MAY be an end-user (natural person) or a non-human entity treated as a person for limited legal purposes (legal person), such as a corporation. A PAT is unique to a resource owner, resource server used for resource management, and authorization server used for protection of those resources. The issuance of the PAT represents the authorization of the resource owner for the resource server to use the authorization server for protecting those resources.

Different grant types for PAT issuance might be appropriate for different types of resource owners; for example, the client credentials grant is useful in the case of an organization acting as a resource owner, whereas an interactive grant type is typically more appropriate for capturing the approval of an end-user resource owner. Where an identity token is desired in addition to an access token, it is RECOMMENDED to use [OIDCCore] in addition.

1.4 Separation of Responsibility and Authority

Federation of authorization for the UMA grant delivers a conceptual separation of responsibility and authority:

- The resource owner can control access to resources residing at multiple resource servers from a single authorization server, by virtue of authorizing PAT issuance for each resource server. Any one resource server MAY be operated by a party different from the one operating the authorization server.
- The resource server defines the boundaries of resources and the scopes available to each resource, and interprets how clients' resource requests map to permission requests, by virtue of being the publisher of the API being protected and using the protection API to communicate to the authorization server.
- The resource owner works with the authorization server to configure policy conditions (authorization grant rules), which the authorization server executes in the process of issuing access tokens. The authorization process makes use of claims gathered from the requesting party and client in order to satisfy all operative operative policy conditions.

The separation of authorization decision making and authorization enforcement is similar to the architectural separation often used in enterprises between policy decision points and policy enforcement points. However, the resource server MAY apply additional authorization controls beyond those imposed by the authorization server. For example, even if an RPT provides sufficient permissions for a particular case, the resource server can choose to bar access based on its own criteria.

Practical control of access among loosely coupled parties typically requires more than just messaging protocols. It is outside the scope of this specification to define more than the technical contract between UMA-conforming entities. Laws may govern authorization-granting relationships. It is RECOMMENDED for the resource owner, authorization server, and resource server to establish agreements about which parties are responsible for establishing and maintaining authorization grant rules and other authorization rules on a legal or contractual level, and parties operating entities claiming to be UMA-conforming should provide documentation of rights and obligations between and among them. See Section 4 of [UMAGrant] for more information.

Except for PAT issuance, the resource owner-resource server and resource owner-authorization server interfaces -- including the setting of policy conditions -- are outside the scope of this specification (see Section 8 and Section 6.1 of [UMAGrant] for privacy considerations). Some elements of the protection API enable the building of user interfaces for policy condition setting (for example, see Section 3.2, which can be used in concert with user interaction for resource protection and sharing and offers an end-user redirection mechanism for policy interactions).

Note: The resource server typically requires access to at least the permission and token introspection endpoints when an end-user resource owner is not available ("offline" access). Thus, the authorization server needs to manage the PAT in a way that ensures this outcome. [UMA-Impl] discusses ways the resource server can enhance its error handling when the PAT is invalid.

1.5 Protection API Summary

The protection API defines the following endpoints:

- Resource registration endpoint as defined in Section 3. The API available at this endpoint provides a means for the resource server to put resources under the protection of an authorization server on behalf of the resource owner and manage them over time.
- Permission endpoint as defined in Section 4. This endpoint provides a means for the resource server to request a set of one or more permissions on behalf of the client based on the client's resource request when that request is unaccompanied by an access token or is accompanied by an RPT that is insufficient for access to that resource.
- OPTIONAL token introspection endpoint as defined in [RFC7662] and as extended in Section 5. This endpoint provides a means for the resource server to introspect the RPT.

Use of these endpoints assumes that the resource server has acquired OAuth client credentials from the authorization server by static or dynamic means, and has a valid PAT. Note: Although the resource identifiers that appear in permission and token introspection request messages could sufficiently identify the resource owner, the

PAT is still required because it represents the resource owner's authorization to use the protection API, as noted in Section 1.3.

The authorization server MUST declare its protection API endpoints in the discovery document (see Section 2).

1.5.1 Permissions

A permission is (requested or granted) authorized access to a particular resource with some number of scopes bound to that resource. The concept of permissions is used in authorization assessment, results calculation, and RPT issuance in [UMAGrant]. This concept takes on greater significance in relation to the protection API.

The resource server's resource registration operations at the authorization server result in a set of resource owner-specific resource identifiers. When the client makes a resource request that is unaccompanied by an access token or its resource request fails, the resource server is responsible for interpreting that request and mapping it to a choice of authorization server, resource owner, resource identifier(s), and set of scopes for each identifier, in order to request one or more permissions -- resource identifiers and a set of scopes -- and obtain a permission ticket on the client's behalf. Finally, when the client has made a resource request accompanied by an RPT and token introspection is in use, the returned token introspection object reveals the structure of permissions, potentially including expiration of individual permissions.

2. Authorization Server Metadata

This specification makes use of the authorization server discovery document structure and endpoint defined in [UMAGrant]. The resource server uses this discovery document to discover the endpoints it needs.

In addition to the metadata defined in that specification and [OAuthMeta], this specification defines the following metadata for inclusion in the discovery document:

permission_endpoint
: REQUIRED. The endpoint URI at which the resource server requests permissions on the client's behalf.

resource_registration_endpoint
: REQUIRED. The endpoint URI at which the resource server registers resources to put them under authorization manager protection.

Following are additional requirements related to metadata:

introspection_endpoint
: If the authorization server supports token introspection as defined in this specification, it MUST supply this metadata value (defined in [OAuthMeta]).

The authorization server SHOULD document any profiled or extended features it supports explicitly, ideally by supplying the URI identifying each UMA profile and extension as an `uma_profiles_supported` metadata array value (defined in [UMAGrant]), and by using extension metadata to indicate specific usage details as necessary.

3. Resource Registration Endpoint

The API available at the resource registration endpoint enables the resource server to put resources under the protection of an authorization server on behalf of the resource owner and manage them over time. Protection of a resource at the authorization server begins on successful registration and ends on successful deregistration.

The resource server uses a RESTful API at the authorization server's resource registration endpoint to create, read, update, and delete resource descriptions, along with retrieving lists of such descriptions. The descriptions consist of JSON documents that are maintained as web resources at the authorization server. (Note carefully the similar but distinct senses in which the word "resource" is used in this section.)

Figure 2 illustrates the resource registration API operations, with requests and success responses shown.

```
authorization              resource         resource
    server                  server           owner
      |                       |                |
      |*PROTECTION API:       |                |
      |*Resource registration |                |
      |endpoint/API           |                |
      |                       |                |
      |*Create resource (POST)|                |
      |<----------------------|                |
      |*201 Created with      |                |
      |resource ID            |                |
      |---------------------->|                |
      |                       |                |
      |Set policy conditions (anytime          |
      |before deletion/deregistration)         |
      |<- - - - - - - - - - - - - - - - - - - -|
      |                       |                |
      |*Read (GET) with       |                |
      |resource ID            |                |
      |<----------------------|                |
      |*200 OK with resource  |                |
      |representation         |                |
      |---------------------->|                |
      |*Update (PUT) with     |                |
      |resource ID            |                |
      |<----------------------|                |
      |*200 OK with resource  |                |
      |ID                     |                |
      |---------------------->|                |
      |*List (GET)            |                |
      |<----------------------|                |
      |*200 OK with list of   |                |
      |resource IDs           |                |
      |---------------------->|                |
      |*Delete (DELETE) with  |                |
      |resource ID            |                |
      |<----------------------|                |
      |*200 OK or 204 No      |                |
      |Content                |                |
      |---------------------->|                |
```

Figure 2: Resource Registration Endpoint and API: Requests and Success Responses

The resource server MAY 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 MAY 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.

The resource server MAY register a single resource for protection that, from its perspective, has multiple parts, or has dynamic elements such as the capacity for querying or filtering, or otherwise has internal complexity. The resource server alone is responsible for maintaining any required mappings between internal representations and the resource identifiers and scopes known to the authorization server.

Note: The resource server is responsible for managing the process and timing of registering resources, maintaining the registration of resources, and deregistering resources at the authorization server. Motivations for updating a resource might include, for example, new scopes added to a new API version or resource owner actions at a resource server that result in new resource description text. See [UMA-Impl] for a discussion of initial resource registration timing options.

3.1 Resource Description

A resource description is a JSON document that describes the characteristics of a resource sufficiently for an authorization server to protect it. A resource description has the following parameters:

resource_scopes   REQUIRED. An array of strings, serving as scope identifiers, indicating the available scopes for this resource. Any of the strings MAY be either a plain string or a URI.

description   OPTIONAL. A human-readable string describing the resource at length. The authorization server MAY use this description in any user interface it presents to a resource owner, for example, for resource protection monitoring or policy setting. The value of this parameter MAY be internationalized, as described in Section 2.2 of [RFC7591].

icon_uri   OPTIONAL. A URI for a graphic icon representing the resource. The authorization server MAY use the referenced icon in any user interface it presents to a resource owner, for example, for resource protection monitoring or policy setting.

name   OPTIONAL. A human-readable string naming the resource. The authorization server MAY use this name in any user interface it presents to a resource owner, for example, for resource protection monitoring or policy setting. The value of this parameter MAY be internationalized, as described in Section 2.2 of [RFC7591].

type   OPTIONAL. A string identifying the semantics of the resource. For example, if the resource is an identity claim that leverages standardized claim semantics for "verified email address", the value of this parameter could be an identifying URI for this claim. The authorization server MAY use this information in processing information about the resource or displaying information about it in any user interface it presents to a resource owner.

For example, this description characterizes a resource (a photo album) that can potentially be viewed or printed; the scope URI points to a scope description as defined in Section 3.1.1:

```
{
   "resource_scopes":[
      "view",
      "http://photoz.example.com/dev/scopes/print"
   ],
   "description":"Collection of digital photographs",
   "icon_uri":"http://www.example.com/icons/flower.png",
   "name":"Photo Album",
   "type":"http://www.example.com/rsrcs/photoalbum"
}
```

3.1.1 Scope Description

A scope description is a JSON document that describes the characteristics of a scope sufficiently for an authorization server to protect the resource with this available scope.

While a scope URI appearing in a resource description (see Section 3.1) MAY resolve to a scope description document, and thus scope description documents are possible to standardize and reference publicly, the authorization server is not expected to resolve scope description details at resource registration time or at any other run-time requirement. The resource server and authorization server are presumed to have negotiated any required interpretation of scope handling out of band.

A scope description has the following parameters:

description   OPTIONAL. A human-readable string describing the resource at length. The authorization server MAY use this description in any user interface it presents to a resource owner, for example, for resource protection monitoring or policy setting. The value of this parameter MAY be internationalized, as described in Section 2.2 of [RFC7591].

icon_uri

OPTIONAL. A URI for a graphic icon representing the scope. The authorization server MAY use the referenced icon in any user interface it presents to a resource owner, for example, for resource protection monitoring or policy setting.

name OPTIONAL. A human-readable string naming the scope. The authorization server MAY use this name in any user interface it presents to a resource owner, for example, for resource protection monitoring or policy setting. The value of this parameter MAY be internationalized, as described in Section 2.2 of [RFC7591].

For example, this scope description characterizes a scope that involves printing (as opposed to, say, creating or editing in some fashion):

```
{
   "description":"Print out and produce PDF files of photos",
   "icon_uri":"http://www.example.com/icons/printer",
   "name":"Print"
}
```

3.2 Resource Registration API

The authorization server MUST support the following five registration options and MUST require a valid PAT for access to them; any other operations are undefined by this specification. Here, *rreguri* stands for the resource registration endpoint and *_id* stands for the authorization server-assigned identifier for the web resource corresponding to the resource at the time it was created, included within the URL returned in the Location header. Each operation is defined in its own section below.

- Create resource description: POST *rreguri/*
- Read resource description: GET *rreguri/_id*
- Update resource description: PUT *rreguri/_id*
- Delete resource description: DELETE *rreguri/_id*
- List resource descriptions: GET *rreguri/*

Within the JSON body of a successful response, the authorization server includes common parameters, possibly in addition to method-specific parameters, as follows:

_id REQUIRED (except for the Delete and List methods). A string value repeating the authorization server-defined identifier for the web resource corresponding to the resource. Its appearance in the body makes it readily available as an identifier for various protected resource management tasks.

user_access_policy_uri OPTIONAL. A URI that allows the resource server to redirect an end-user resource owner to a specific user interface within the authorization server where the resource owner can immediately set or modify access policies subsequent to the resource registration action just completed. The authorization server is free to choose the targeted user interface, for example, in the case of a deletion action, enabling the resource server to direct the end-user to a policy-setting interface for an overall "folder" resource formerly "containing" the deleted resource (a relationship the authorization server is not aware of), to enable adjustment of related policies.

If the request to the resource registration endpoint is incorrect, then the authorization server instead responds as follows (see Section 6 for information about error messages):

- If the referenced resource cannot be found, the authorization server MUST respond with an HTTP 404 (Not Found) status code and MAY respond with a `not_found` error code.
- If the resource server request used an unsupported HTTP method, the authorization server MUST respond with the HTTP 405 (Method Not Allowed) status code and MAY respond with an `unsupported_method_type` error code.
- If 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 and MAY respond with an `invalid_request` error code.

3.2.1 Create Resource Description

Adds a new resource description to the authorization server using the POST method. If the request is successful, the resource is thereby registered and the authorization server MUST respond with an HTTP 201 status message that includes a `Location` header and an `_id` parameter.

Form of a create request, with a PAT in the header:

```
POST /rreg/ HTTP/1.1 Content-Type: application/json
Authorization: Bearer MHg3OUZEQkZBMjcx
...
{
   "resource_scopes":[
      "read-public",
      "post-updates",
      "read-private",
      "http://www.example.com/scopes/all"
   ],
   "icon_uri":"http://www.example.com/icons/sharesocial.png",
   "name":"Tweedl Social Service",
   "type":"http://www.example.com/rsrcs/socialstream/140-compatible"
}
```

Form of a successful response, also containing an optional `user_access_policy_uri` parameter:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: /rreg/KX3A-39WE
...
{
   "_id":"KX3A-39WE",
   "user_access_policy_uri":"http://as.example.com/rs/222/resource/KX3A-39WE/policy"
}
```

3.2.2 Read Resource Description

Reads a previously registered resource description using the GET method. If the request is successful, the authorization server MUST respond with an HTTP 200 status message that includes a body containing the referenced resource description, along with an `_id` parameter.

Form of a read request, with a PAT in the header:

```
GET /rreg/KX3A-39WE HTTP/1.1
Authorization: Bearer MHg3OUZEQkZBMjcx
...
```

Form of a successful response, containing all the parameters that were registered as part of the description:

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
   "_id":"KX3A-39WE",
   "resource_scopes":[
      "read-public",
      "post-updates",
      "read-private",
      "http://www.example.com/scopes/all"
   ],
   "icon_uri":"http://www.example.com/icons/sharesocial.png",
   "name":"Tweedl Social Service",
   "type":"http://www.example.com/rsrcs/socialstream/140-compatible"
}
```

3.2.3 Update Resource Description

Updates a previously registered resource description, by means of a complete replacement of the previous resource description, using the PUT method. If the request is successful, the authorization server MUST respond with an HTTP 200 status message that includes an `_id` parameter.

Form of an update request adding a `description` parameter to a resource description that previously had none, with a PAT in the header:

```
PUT /rreg/9UQU-DUWW HTTP/1.1
Content-Type: application/json
Authorization: Bearer 204c69636b6c69
...
{
   "resource_scopes":[
      "http://photoz.example.com/dev/scopes/view",
      "public-read"
   ],
   "description":"Collection of digital photographs",
   "icon_uri":"http://www.example.com/icons/sky.png",
   "name":"Photo Album",
   "type":"http://www.example.com/rsrcs/photoalbum"
}
```

Form of a successful response, not containing the optional `user_access_policy_uri` parameter:

```
HTTP/1.1 200 OK
...
{
   "_id":"9UQU-DUWW"
}
```

3.2.4 Delete Resource Description

Deletes a previously registered resource description using the DELETE method. If the request is successful, the resource is thereby deregistered and the authorization server MUST respond with an HTTP 200 or 204 status message.

Form of a delete request, with a PAT in the header:

```
DELETE /rreg/9UQU-DUWW
Authorization: Bearer 204c69636b6c69
...
```

Form of a successful response:

```
HTTP/1.1 204 No content
...
```

3.2.5 List Resource Descriptions

Lists all previously registered resource identifiers for this resource owner using the GET method. The authorization server MUST return the list in the form of a JSON array of `_id` string values.

The resource server can use this method as a first step in checking whether its understanding of protected resources is in full synchronization with the authorization server's understanding.

Form of a list request, with a PAT in the header:

```
GET /rreg/ HTTP/1.1
Authorization: Bearer 204c69636b6c69
...
```

Form of a successful response:

```
HTTP/1.1 200 OK
...
[
   "KX3A-39WE",
   "9UQU-DUWW"
]
```

4. Permission Endpoint

The permission endpoint defines a means for the resource server to request one or more permissions (resource identifiers and corresponding scopes) with the authorization server on the client's behalf, and to receive a permission ticket in return, in order to respond as indicated in Section 3.2 of [UMAGrant]. The resource server uses this endpoint on the following occasions:

- After the client's initial resource request without an access token
- After the client's resource request that was accompanied by an invalid RPT or a valid RPT that had insufficient permissions associated with it

The use of the permission endpoint is illustrated in Figure 3, with a request and a success response shown.

```
               authorization          resource
client             server               server
  |                  |                     |
  |Request resource (no or insufficient    |
  |access token)     |                     |
  |--------------------------------------->|
  |                  |                     |
  |                  |*PROTECTION API:     |
  |                  |*Permission endpoint |
  |                  |                     |
  |                  |*Request permissions |
  |                  |(POST)               |
  |                  |<--------------------|
  |                  |*201 Created with    |
  |                  |permission ticket    |
  |                  |-------------------->|
  |                  |                     |
  |401 response with permission ticket,    |
  |authz server location                   |
  |<---------------------------------------|
```

Figure 3: Permission Endpoint: Request and Success Response

The PAT provided in the API request enables the authorization server to map the resource server's request to the appropriate resource owner. It is only possible to request permissions for access to the resources of a single resource owner, protected by a single authorization server, at a time.

In its response, the authorization server returns a permission ticket for the resource server to give to the client that represents the same permissions that the resource server requested.

The process of choosing what permissions to request from the authorization server may require interpretation and mapping of the client's resource request. The resource server SHOULD request a set of permissions with scopes that is reasonable for the client's resource request. The resource server MAY request multiple permissions, and any permission MAY have zero scopes associated with it. Requesting multiple permissions might be appropriate, for example, in cases where the resource server expects the requesting party to need access to several related resources if they need access to any one of the resources (see Section 3.3.4 of [UMAGrant] for an example). Requesting a permission with no scopes might be appropriate, for example, in cases where an access attempt involves an API call that is ambiguous without further context (role-based scopes such as `user` and `admin` could have this ambiguous quality, and an explicit client request for a particular scope at the token endpoint later can clarify the desired access). The resource server SHOULD document its intended pattern of permission requests in order to assist the client in pre-registering for and requesting appropriate scopes at the authorization server. See [UMA-Impl] for a discussion of permission request patterns.

Note: In order for the resource server to know which authorization server to approach for the permission ticket and on which resource owner's behalf (enabling a choice of permission endpoint and PAT), it needs to derive the necessary information using cues provided by the structure of the API where the resource request was made, rather than by an access token. Commonly, this information can be passed through the URI, headers, or body of the client's request. Alternatively, the entire interface could be dedicated to the use of a single resource owner and protected by a single authorization server.

4.1 Resource Server Request to Permission Endpoint

The resource server uses the POST method at the permission endpoint. The body of the HTTP request message contains a JSON object for requesting a permission for single resource identifier, or an array of one or more objects for requesting permissions for a corresponding number of resource identifiers. The object format in both cases is derived from the resource description format specified in Section 3.1; it has the following parameters:

resource_id REQUIRED. The identifier for a resource to which the resource server is requesting a permission on behalf of the client. The identifier MUST correspond to a resource that was previously registered.

resource_scopes REQUIRED. An array referencing zero or more identifiers of scopes to which the resource server is requesting access for this resource on behalf of the client. Each scope identifier MUST correspond to a scope that was previously registered by this resource server for the referenced resource.

Example of an HTTP request for a single permission at the authorization server's permission endpoint, with a PAT in the header:

```
POST /perm HTTP/1.1
Content-Type: application/json
Host: as.example.com
Authorization: Bearer 204c69636b6c69
...

{
   "resource_id":"112210f47de98100",
   "resource_scopes":[
      "view",
      "http://photoz.example.com/dev/actions/print"
   ]
}
```

Example of an HTTP request for multiple permissions at the authorization server's permission endpoint, with a PAT in the header:

```
POST /perm HTTP/1.1
Content-Type: application/json
Host: as.example.com
Authorization: Bearer 204c69636b6c69
...

[
   {
      "resource_id":"7b727369647d",
      "resource_scopes":[
         "view",
         "crop",
         "lightbox"
      ]
   },
   {
      "resource_id":"7b72736964327d",
      "resource_scopes":[
         "view",
         "layout",
         "print"
      ]
   },
   {
      "resource_id":"7b72736964337d",
      "resource_scopes":[
         "http://www.example.com/scopes/all"
      ]
   }
]
```

4.2 Authorization Server Response to Resource Server on Permission Request Success

If the authorization server is successful in creating a permission ticket in response to the resource server's request, it responds with an HTTP 201 (Created) status code and includes the `ticket` parameter in the JSONformatted body. Regardless of whether the request contained one or multiple permissions, only a single permission ticket is returned.

For example:

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

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

4.3 Authorization Server Response to Resource Server on Permission Request Failure

If the resource server's permission 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 error codes (see Section 6 for more information about error codes and responses):

invalid_resource_id   At least one of the provided resource identifiers 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 for the referenced resource.

5. Token Introspection Endpoint

When the client makes a resource request accompanied by an RPT, the resource server needs to determine whether the RPT is active and, if so, its associated permissions. Depending on the nature of the RPT and operative caching parameters, the resource server MAY take any of the following actions as appropriate to determine the RPT's status:

- Introspect the RPT at the authorization server using the OAuth token introspection endpoint (defined in [RFC7662] and this section) that is part of the protection API. The authorization server's response contains an extended version of the introspection response. If the authorization server supports this specification's version of the token introspection endpoint, it MUST declare the endpoint in its discovery document (see Section 2) and support this extended version of the response.
- Use a cached copy of the token introspection response if allowed (see Section 4 of [RFC7662]).
- Validate the RPT locally if it is self-contained.

The use of the token introspection endpoint is illustrated in Figure 4, with a request and a success response shown.

```
               authorization              resource
client             server                  server
  |                  |                       |
  |Resource request with RPT                 |
  |----------------------------------------->|
  |                  |                       |
  |                  |*PROTECTION API:       |
  |                  |*Introspection endpoint|
  |                  |                       |
  |                  |*Request to introspect |
  |                  |token (POST)           |
  |                  |<----------------------|
  |                  |*Response with token   |
  |                  |introspection object   |
  |                  |---------------------->|
  |                  |                       |
  |Protected resource                        |
  |<-----------------------------------------|
```

Figure 4: Token Introspection Endpoint: Request and Success Response

The authorization server MAY support both UMA-extended and non-UMA introspection requests and responses.

5.1 Resource Server Request to Token Introspection Endpoint

Note: In order for the resource server to know which authorization server, PAT (representing a resource owner), and endpoint to use in making the token introspection API call, it may need to interpret the client's resource request.

Example of the resource server's request to the authorization server for introspection of an RPT, with a PAT in the header:

```
POST /introspect HTTP/1.1
Host: as.example.com
Authorization: Bearer 204c69636b6c69
...
token=sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv
```

Because an RPT is an access token, if the resource server chooses to supply a token type hint, it would use a `token_type_hint` of `access_token`.

5.1.1 Authorization Server Response to Resource Server on Token Introspection Success

The authorization server's response to the resource server MUST use [RFC7662], responding with a JSON object with the structure dictated by that specification, extended as follows.

If the introspection object's `active` parameter has a Boolean value of `true`, then the object MUST NOT contain a `scope` parameter, and MUST contain an extension parameter named `permissions` that contains an array of objects, each one (representing a single permission) containing these parameters:

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

resource_scopes REQUIRED. An array referencing zero or more strings representing scopes to which access was granted for this resource. Each string MUST correspond to a scope that was registered by this resource server for the referenced resource.

exp OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this permission will expire. If the token-level `exp` value pre-dates a permission-level `exp` value, the token-level value takes precedence.

iat OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this permission was originally issued. If the token-level `iat` value post-dates a permission-level `iat` value, the token-level value takes precedence.

nbf OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating the time before which this permission is not valid. If the token-level `nbf` value post-dates a permission-level `nbf` value, the token-level value takes precedence.

Example of a response containing the introspection object with the `permissions` parameter containing a single permission:

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

{
   "active":true,
   "exp":1256953732,
   "iat":1256912345,
   "permissions":[
      {
         "resource_id":"112210f47de98100",
         "resource_scopes":[
            "view",
            "http://photoz.example.com/dev/actions/print"
         ],
         "exp":1256953732
      }
   ]
}
```

6. Error Messages

If a request is successfully authenticated, but is invalid for another reason, the authorization server produces an error response by supplying a JSON-encoded object with the following members in the body of the HTTP response:

error   REQUIRED except as noted. A single error code. Values for this parameter 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.

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

{
  "error": "invalid_resource_id",
  "error_description": "Permission request failed with bad resource ID.",
  "error_uri": "https://as.example.com/uma_errors/invalid_resource_id"
}
```

7. Security Considerations

This specification inherits the security considerations of [UMAGrant] and has the following additional security considerations.

In the context of federated authorization, more parties may be operating and using UMA software entities, and thus may need to establish agreements about the parties' rights and responsibilities on a legal or contractual level, as discussed in Section 5.8 of [UMAGrant].

The protection API is secured by means of OAuth (through the use of the PAT). Therefore, it is susceptible to OAuth threats.

8. Privacy Considerations

This specification inherits the privacy considerations of [UMAGrant] and has the following additional privacy considerations.

As noted in Section 6.1 of [UMAGrant], the authorization server should apply authorization, security, and time-to-live strategies in a way that favors resource owner needs and action so that removal of authorization grants is achieved in a timely fashion. PATs are another construct to which it can apply these strategies.

In the context of federated authorization, more parties may be operating and using UMA software entities, and thus may need to establish agreements about mutual rights, responsibilities, and common interpretations of UMA constructs for consistent and expected software behavior, as discussed in Section 6.4 of [UMAGrant].

The authorization server comes to be in possession of resource details that may reveal information about the resource owner, which the authorization server's trust relationship with the resource server is assumed to accommodate. The more information about a resource that is registered, the more risk of privacy compromise there is through a less-trusted authorization server. For example, if resource owner Alice introduces her electronic health record resource server to an authorization server in the cloud, the authorization server may come to learn a great deal of detail about Alice's health information just so that she can control access by others to that information.

9. IANA Considerations

This document makes the following requests of IANA.

9.1 OAuth 2.0 Authorization Server Metadata Registry

This specification registers OAuth 2.0 authorization server metadata defined in Section 2, as required by Section 7.1 of [OAuthMeta].

9.1.1 Registry Contents

- Metadata name: `permission_endpoint`
- Metadata description: endpoint metadata
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 2 in this document

- Metadata name: `resource_registration_endpoint`
- Metadata description: endpoint metadata
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 2 in this document

9.2 OAuth Token Introspection Response Registration

This specification registers the name defined in Section 5.1.1, as required by Section 3.1 of [RFC7662].

9.2.1 Registry Contents

- Name: `permissions`
- Description: array of objects, each describing a scoped, time-limitable permission for a resource
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 5.1.1 in this document

10. Acknowledgments

The following people made significant text contributions to the specification:

- Paul C. Bryan, ForgeRock US, Inc. (former editor)
- Domenico Catalano, Oracle (former author)
- Mark Dobrinic, Cozmanova
- George Fletcher, AOL
- Thomas Hardjono, MIT (former editor)
- Andrew Hindle, Hindle Consulting Limited
- Lukasz Moren, Cloud Identity Ltd
- James Phillpotts, ForgeRock
- Christian Scholz, COMlounge GmbH (former editor)
- Mike Schwartz, Gluu
- Cigdem Sengul, Nominet UK
- 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].

11. References

11.1 Normative References

[BCP195] Sheffer, Y., “Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)”, May 2015, <https://tools.ietf.org/html/bcp195>.

[OIDCCore] Sakimura, N., “OpenID Connect Core 1.0 incorporating errata set 1”, November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[UMAGrant] Maler, E., “User-Managed Access (UMA) Grant for OAuth 2.0 Authorization”, January 2018, <https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html>.

[OAuthMeta] Jones, M., “OAuth 2.0 Authorization Server Metadata”, November 2017, <https://tools.ietf.org/html/draft-ietf-oauth-discovery-08>.

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6750] Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage”, RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC6749] Hardt, D., Ed., “The OAuth 2.0 Authorization Framework”, RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC7159] Bray, T., Ed., “The JavaScript Object Notation (JSON) Data Interchange Format”, RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, “OAuth 2.0 Dynamic Client Registration Protocol”, RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[RFC7662] Richer, J., Ed., “OAuth 2.0 Token Introspection”, RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

11.2 Informative References

[UMA-Impl] Maler, E., “UMA Implementer's Guide”, 2017, <https://kantarainitiative.org/confluence/display/uma/UMA+Implementer%27s+Guide>.

[UMAnitarians] Maler, E., “UMA Participant Roster”, 2017, <https://kantarainitiative.org/confluence/display/uma/Participant+Roster>.

Authors' Addresses

Eve Maler (editor)
ForgeRock
EMail: eve.maler@forgerock.com

Maciej Machulak
HSBC
EMail: maciej.p.machulak@hsbc.com

Justin Richer
Bespoke Engineering
EMail: justin@bspk.io