User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization

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 a client, representing a requesting party, to use a permission ticket to request an OAuth 2.0 access token to gain access to a protected resource asynchronously from the time a resource owner authorizes access.

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 defines an extension OAuth 2.0 [RFC6749] grant. The grant enhances OAuth capabilities in the following ways:

- The resource owner authorizes protected resource access to clients used by entities that are in a *requesting party* role. This enables party-to-party authorization, rather than authorization of application access alone.
- The authorization server and resource server interact with the client and requesting party in a way that is *asynchronous* with respect to resource owner interactions. This lets a resource owner configure an authorization server with authorization grant rules (policy conditions) at will, rather than authorizing access token issuance synchronously just after authenticating.

For example, bank customer (resource owner) Alice with a bank account service (resource server) can use a sharing management service (authorization server) hosted by the bank to manage access to her various protected resources by spouse Bob, accounting professional Charline, and and financial information aggregation company Decide Account, all using different client applications. Each of her bank accounts is a protected resource, and two different scopes of access she can control on them are viewing account data and accessing payment functions.

An OPTIONAL second specification, [UMAFedAuthz], defines a means for an UMA-enabled authorization server and resource server to be loosely coupled, or federated, in a resource owner context. This specification, together with [UMAFedAuthz], constitutes UMA 2.0.

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 Roles

The UMA grant enhances the OAuth definitions of entities in order to accommodate the requesting party role.

resource owner
: An entity capable of granting access to a protected resource, the "user" in User-Managed Access. The resource owner MAY be an end-user (natural person) or MAY be a non-human entity treated as a person for limited legal purposes (legal person), such as a corporation.

requesting party
: A natural or 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 that is capable of making requests for protected resources with the resource owner's authorization and on the requesting party's behalf.

resource server
: A server that hosts resources on a resource owner's behalf and is capable of accepting and responding to requests for protected resources.

authorization server
: A server that protects, on a resource owner's behalf, resources hosted at a resource server.

1.3 Abstract Flow

The UMA grant enhances the abstract protocol flow of OAuth.

Figure 1 shows an example flow illustrating a variety of messaging paths and artifacts. The resource owner entity and its communications with the authorization server are included for completeness, although policy condition setting is outside the scope of this specification and communications among the other four entities are asynchrjonous with respect to resource owner actions. Further, although both claims pushing and interactive claims gathering are shown, both might not typically be used in one scenario.

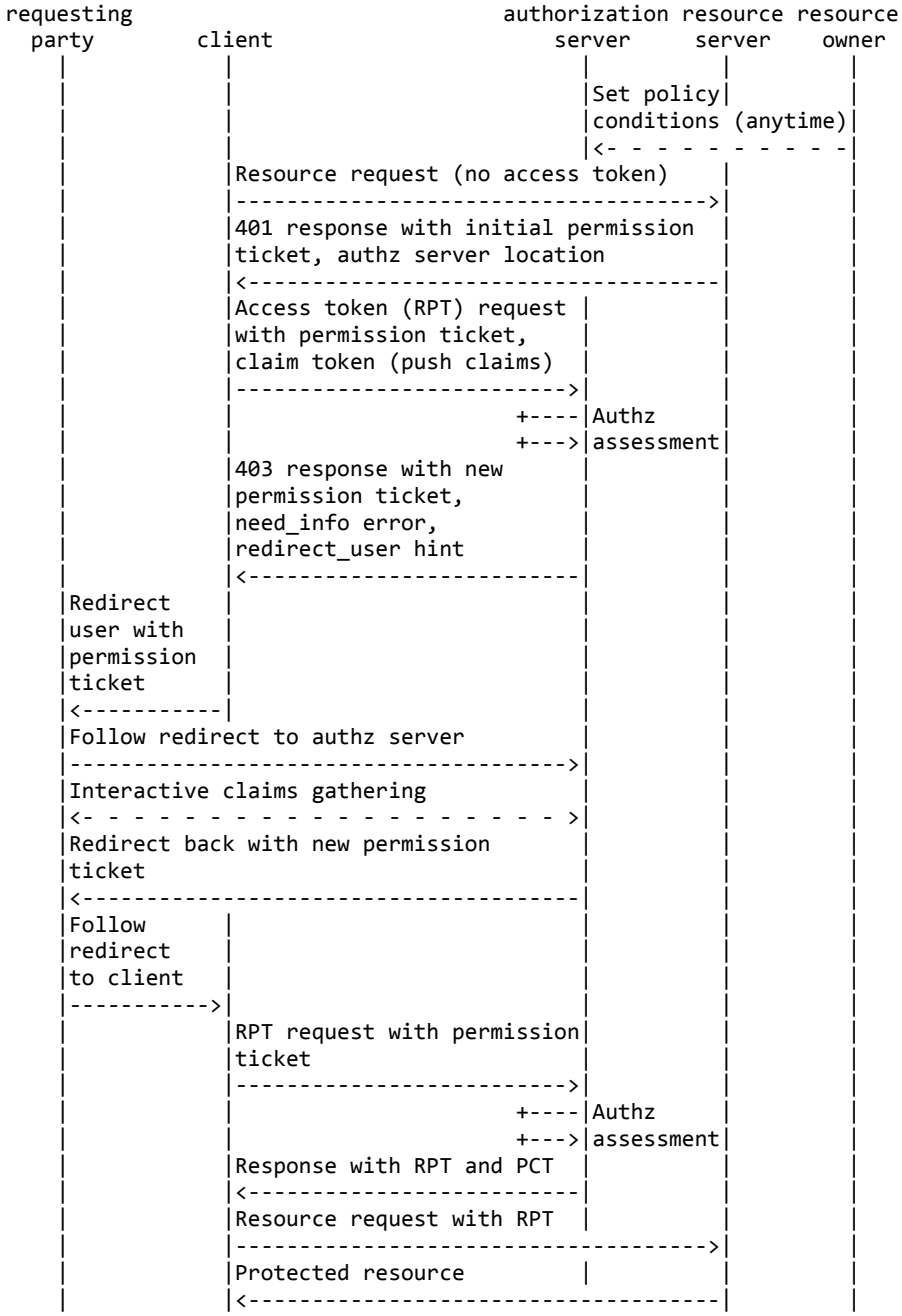

Figure 1: Example Flow

Following are key concepts relevant to this specification, as illustrated in the figure:

requesting party token (RPT) An OAuth access token associated with the UMA grant. An RPT is unique to a requesting party, client, authorization server, resource server, and resource owner.

permission Authorized access to a particular resource with some number of scopes bound to that resource. A permission ticket represents some number of requested permissions. An RPT represents some number of granted permissions. Permissions are part of the authorization server's process and are opaque to the client.

permission ticket A correlation handle representing requested permissions that is created and maintained by the authorization server, initially passed to the client by the resource server, and presented by the client at the token endpoint and during requesting party redirects.

authorization process The process through which the authorization server determines whether it should issue an RPT to the client on the requesting party's behalf, based on a variety of inputs. A key component of the process is authorization assessment. (See Section 1.3.1.)

claim A statement of the value or values of one or more attributes of an entity. The authorization server typically needs to collect and assess one or more claims of the requesting party or client against policy conditions as part of protecting a resource. The two methods available for UMA claims collection are claims pushing and interactive claims gathering. Note: Claims collection might involve authentication for unique user identification, but depending on policy conditions might additionally or instead involve the collection of non-uniquely identifying attributes, authorization for some action (for example, see Section 3.3.3), or other statements of agreement.

claim token A package of claims provided directly by the client to the authorization server through claims pushing.

persisted claims token (PCT) A correlation handle issued by an authorization server that represents a set of claims collected during one authorization process, available for a client to use in attempting to optimize a future authorization process.

Note: How the client acquired knowledge of the resource server's interface and the specific endpoint of the desired protected resource is outside the scope of this specification. For example, the resource server might have a programmatic API or it might serve up simple web pages, and the resource owner might have advertised the endpoint publicly on a blog or other website, listed it in a discovery service, or emailed a link to a particular intended requesting party.

1.3.1 Authorization Process

The authorization process involves the following activities:

- Claims collection. Claims pushing by a client is defined in Section 3.3.1, and interactive claims gathering with an end-user requesting party is defined in Section 3.3.2.
- Authorization assessment (as defined in Section 3.3.4). Authorization assessment involves the authorization server assembling and evaluating policy conditions, scopes, claims, and any other relevant information sourced outside of UMA claims collection flows, in order to mitigate access authorization risk.
- Authorization results determination (as defined in Section 3.3.4). The authorization server either returns a success code (as defined in Section 3.3.5), an RPT, and an optional PCT, or an error code (as defined in Section 3.3.6). If the error code is `need_info` or `request_submitted`, the authorization server provides a permission ticket, giving the client an opportunity to continue within the same authorization process (including engaging in further claims collection).

Different choices of claims collection methods, other inputs to authorization assessment, and error codes might be best suited for different deployment ecosystems. For example, where no pre-established relationship is expected between the resource owner's authorization server and the requesting party, initial requesting party redirection might be a useful pattern, at which point the authorization server might either authenticate the requesting party locally or serve as a relying party for a remote identity provider. Where a common authorization server functions as an identity provider for all resource owners and requesting parties, having the client push claim tokens sourced from that central server itself with a pre-negotiated format and contents might be a useful pattern.

2. Authorization Server Metadata

The authorization server supplies metadata in a discovery document to declare its endpoints. The client uses this discovery document to discover these endpoints for use in the flows defined in Section 3.

The authorization server MUST make a discovery document available. The structure of the discovery document MUST conform to that defined in [OAuthMeta]. The discovery document MUST be available at an endpoint formed by concatenating the string `/.well-known/uma2-configuration` to the `issuer` metadata value defined in [OAuthMeta], using the well-known URI syntax and semantics defined in [RFC5785]. In addition to the metadata defined in [OAuthMeta], this specification defines the following metadata for inclusion in the discovery document:

claims_interaction_endpoint
: OPTIONAL. A static endpoint URI at which the authorization server declares that it interacts with end-user requesting parties to gather claims. If the authorization server also provides a claims interaction endpoint URI as part of its `redirect_user` hint in a `need_info` response to a client on authorization failure (see Section 3.3.6), that value overrides this metadata value. Providing the static endpoint URI is useful for enabling interactive claims gathering prior to any pushed-claims flows taking place, for example, for gathering authorization for subsequent claim pushing (see Section 3.3.2).

uma_profiles_supported
: OPTIONAL. UMA profiles and extensions supported by this authorization server. The value is an array of string values, where each string value is a URI identifying an UMA profile or extension. As discussed in Section 4, an authorization server supporting a profile or extension related to UMA SHOULD supply the specification's identifying URI (if any) here.

If the authorization server supports dynamic client registration, it MUST allow client applications to register `claims_redirect_uri` metadata, as defined in Section 3.3.2, using the following metadata field:

claims_redirect_uris
: OPTIONAL. Array of one or more claims redirection URIs.

3. Flow Details

3.1 Client Requests Resource Without Providing an Access Token

The client requests a protected resource without providing any access token.

Note: This process does not assume that any relevant policy conditions have already been defined at the authorization server.

For an example of how the resource server can put resources under the protection of an authorization server, see [UMAFedAuthz].

Example of a client request at a protected resource without providing an access token:

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

3.2 Resource Server Responds to Client's Tokenless Access Attempt

The resource server responds to the client's tokenless resource request.

The resource server MUST obtain a permission ticket from the authorization server to provide in its response, but the means of doing so is outside the scope of this specification. For an example of how the resource server can obtain the permission ticket, see [UMAFedAuthz].

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.

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, 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.

See Section 5.5 for permission ticket security considerations.

3.2.1 Resource Server Response to Client on Permission Request Success

If the resource server is able to provide a permission ticket from the authorization server, it responds to the client by providing a `WWW-Authenticate` header with the authentication scheme `UMA`, with the `issuer` URI from the authorization server's discovery document in an `as_uri` parameter and the permission ticket in a `ticket` parameter.

For example:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="example",
  as_uri="https://as.example.com",
  ticket="016f84e8-f9b9-11e0-bd6f-0021cc6004de"
...
```

3.2.2 Resource Server Response to Client on Permission Request Failure

If the resource server is unable to provide a permission ticket from the authorization server, then it includes a header of the following form in its response to the client: `Warning: 199 - "UMA Authorization Server Unreachable"`.

For example:

```
HTTP/1.1 403 Forbidden
Warning: 199 - "UMA Authorization Server Unreachable"
...
```

Without an authorization server location and permission ticket, the client is unable to continue.

3.3 Client Seeks RPT on Requesting Party's Behalf

The client seeks issuance of an RPT.

This process assumes that:

- The client has obtained a permission ticket and an authorization server location from the resource server.
- The client has retrieved the authorization server's discovery document as needed.
- The client has obtained a client identifier or a full set of client credentials as appropriate, either statically or dynamically (for example, through [RFC7591] or [OIDCDynClientReg]). This grant works with clients of both confidential and public types.

Initiation of this process has two options. One option is for the client to request an RPT from the token endpoint immediately, as defined in Section 3.3.1. Claim pushing is available at this endpoint. The other option, if the authorization server's discovery document statically provided a claims interaction endpoint, is for the client to redirect the requesting party immediately to that endpoint for interactive claims gathering, as defined in Section 3.3.2.

3.3.1 Client Request to Authorization Server for RPT

The client makes a request to the token endpoint by sending the following parameters:

grant_type REQUIRED. MUST be the value `urn:ietf:params:oauth:grant-type:uma-ticket`.

ticket REQUIRED. The most recent permission ticket received by the client as part of this authorization process.

claim_token OPTIONAL. If this parameter is used, it MUST appear together with the `claim_token_format` parameter. A string containing directly pushed claim information in the indicated format. It MUST be base64url encoded unless specified otherwise by the claim token format. The client MAY provide this information on both first and subsequent requests to this endpoint. The client and authorization server together might need to establish proper audience restrictions for the claim token prior to claims pushing. See Section 5.7 and Section 6.2 for security and privacy considerations regarding pushing of claims.

claim_token_format OPTIONAL. If this parameter is used, it MUST appear together with the `claim_token` parameter. A string specifying the format of the claim token in which the client is directly pushing claims to the authorization server. The string MAY be a URI. Examples of potential types of claim token formats are [OIDCCore] ID Tokens and SAML assertions.

pct OPTIONAL. If the authorization server previously returned a PCT along with an RPT, the client MAY include the PCT in order to optimize the process of seeking a new RPT. Given that some claims represented by a PCT are likely to contain identity information about a requesting party, a client supplying a PCT in its RPT request MUST make a best effort to ensure that the requesting party using the client now is the same as the requesting party that was associated with the PCT when it was issued. See Section 5.7 and Section 6.2 for additional security and privacy considerations regarding persistence of claims. The client MAY use the PCT for the same requesting party when seeking an RPT for a resource different from the one sought when the PCT was issued, or a protected resource at a different resource server entirely. See Section 5.2 for additional PCT security considerations. See Section 3.3.5 for

the form of the authorization server's response with a PCT.

rpt OPTIONAL. Supplying an existing RPT (which MAY be expired) gives the authorization server the option of upgrading that RPT instead of issuing a new one (see Section 3.3.5.1 for more about this option).

scope OPTIONAL. A string of space-separated values representing requested scopes. For the authorization server to consider any requested scope in its assessment, the client MUST have been pre-registered for the same scope with the authorization server. The client should consult the resource server's API documentation for details about which scopes it can expect the resource server's initial returned permission ticket to represent as part of the authorization assessment (see Section 3.3.4).

Example of a request message with no optional parameters (line breaks are shown only for display convenience):

```
POST /token HTTP/1.1
Host: as.example.com
Authorization: Basic jwfLG53^sad$#f
...
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Auma-ticket
&ticket=016f84e8-f9b9-11e0-bd6f-0021cc6004de
```

Example of a request message that includes an existing RPT for upgrading, a scope being sought that was previously registered with the authorization server, and a PCT and a claim token for consideration in the authorization process:

```
POST /token HTTP/1.1
Host: as.example.com
Authorization: Basic jwfLG53^sad$#f
...
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Auma-ticket
&ticket=016f84e8-f9b9-11e0-bd6f-0021cc6004de
&claim_token=eyj0...
&claim_token_format=http%3A%2F%2Fopenid.net%2Fspecs%2Fopenid-connect-core-1_0.html%23IDToken
&pct=c2F2ZWRjb25zZW50
&rpt=sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv
&scope=read
```

This specification provides a means to define profiles of claim token formats for use with UMA (see Section 4). The authorization server SHOULD document the profiles it supports in its discovery document.

3.3.2 Client Redirect of Requesting Party to Authorization Server for Interactive Claims-Gathering

The client redirects an end-user requesting party to the authorization server's claims interaction endpoint for one or more interactive claims-gathering processes as the authorization server requires. These can include direct interactions, such as account registration and authentication local to the authorization server as an identity provider, filling out a questionnaire, or asking the user to authorize subsequent collection of claims by interaction or pushing, and persistent storage of such claims (for example, as associated with a PCT). Interactions could also involve further redirection, for example, for federated (such as social) authentication at a remote identity provider, and other federated claims gathering. See Section 5.7 and Section 6.2 for security and privacy considerations regarding pushing and persistence of claims.

The client might have initiated redirection immediately on receiving an initial permission ticket from the resource server, or, for example, in response to receiving a `redirect_user` hint in a `need_info` error (see Section 3.3.6).

In order for the client to redirect the requesting party immediately on receiving the initial permission ticket from the resource server, this process assumes that the authorization server has statically declared its claims interaction endpoint in its discovery document.

The client constructs the request URI by adding the following parameters to the query component of the claims interaction endpoint URI using the `application/x-www-form-urlencoded` format:

client_id REQUIRED. The client's identifier issued by the authorization server.

ticket REQUIRED. The most recent permission ticket received by the client as part of this authorization process.

claims_redirect_uri REQUIRED if the client has pre-registered multiple claims redirection URIs or has pre-registered no claims redirection URI; OPTIONAL only if the client has pre-registered a single claims redirection URI. The URI to which the client wishes the authorization server to direct the requesting party's user agent after completing its interaction. The URI MUST be absolute, MAY contain an `application/x-www-form-urlencoded`-formatted query parameter component that MUST be retained when adding additional parameters, and MUST NOT contain a fragment component. The client SHOULD pre-register its `claims_redirect_uri` with the authorization server, and the authorization server SHOULD require all clients, and MUST require public clients, to pre-register their claims redirection endpoints (see Section 2). Claims redirection URIs are different from the redirection URIs defined in [RFC6749] in that they are intended for the exclusive use of requesting parties and not resource owners. Therefore, authorization servers MUST NOT redirect requesting parties to pre-registered redirection URIs defined in [RFC6749] unless such URIs are also pre-registered specifically as claims redirection URIs. If the URI is pre-registered, this URI MUST exactly match one of the pre-registered claims redirection URIs, with the matching performed as described in Section 6.2.1 of [RFC3986] (Simple String Comparison).

state RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user agent back to the client. The use of this parameter is for preventing cross-site request forgery (see Section 5.1 for further security information).

Example of a request issued by a client application (line breaks are shown only for display convenience):

```
GET /rqp_claims?client_id=some_client_id
&ticket=016f84e8-f9b9-11e0-bd6f-0021cc6004de
&claims_redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fredirect_claims
&state=abc HTTP/1.1
Host: as.example.com
```

3.3.3 Authorization Server Redirect of Requesting Party Back to Client After Interactive Claims-Gathering

At the conclusion of a successful interaction with the requesting party, the authorization server returns the requesting party to the client, adding the following parameters to the query component of the claims redirection URI using the `application/x-www-form-urlencoded` format:

ticket REQUIRED. A permission ticket that allows the client to make further requests to the authorization server during this authorization process. The value MUST NOT be the same as the one the client used to make its request.

state OPTIONAL. The same state value that the client provided in the request. It MUST be present if and only if the client provided it (see Section 5.1 for further security information).

Note: Interactive claims-gathering processes are outside the scope of this specification. The purpose of the interaction is for the authorization server to gather information for its own authorization assessment purposes. This redirection does not involve sending any of the information back to the client.

The authorization server MAY use interactive claims-gathering to request authorization from the requesting party for persisting claims across authorization processes. Such persisted claims will be represented by a PCT issued to the client in a subsequent step.

The client MUST ignore unrecognized response parameters. If the request fails due to a missing, invalid, or mismatching claims redirection URI, or if the client identifier is missing or invalid, the authorization server SHOULD inform the requesting party of the error and MUST NOT automatically redirect the user agent to the invalid redirection URI.

If the request fails for reasons other than a missing or invalid claims redirection URI, the authorization server informs the client by adding an `error` parameter to the query component of the claims redirection URI as defined in Section 4.1.2.1 of

[RFC6749].

Example of a response issued by an authorization server (line breaks are shown only for display convenience):

```
HTTP/1.1 302 Found
Location: https://client.example.com/redirect_claims?
ticket=cHJpdmFjeSBpcyBjb250ZXh0LCBjb250cm9s&state=abc
```

3.3.4 Authorization Assessment and Results Determination

When the authorization server has received a request for an RPT from a client as defined in Section 3.3.1, it assesses whether the client is authorized to receive the requested RPT and determines the results.

The authorization server MUST apply the following conceptual authorization assessment calculation in determining authorization results. Note: As this calculation is internal to authorization server operations, its particulars are outside the scope of this specification.

1. Assemble a set called *RegisteredScopes* containing the scopes for which the client is pre-registered (either dynamically or through some static process) at the authorization server. Assemble a set called *RequestedScopes* containing the scopes the client most recently requested at the token endpoint. The permission ticket that was presented by the client at the token endpoint represents some number of resources, each with some number of scopes; for each of those resources, assemble a set called *TicketScopes(resource)* containing the scopes associated with that resource.
2. For each resource in the permission ticket, determine a final set of requested scopes as follows: *RequestedScopes(resource)={TicketScopes(resource)* $\cup$ *{RegisteredScopes* $\cap$ *RequestedScopes}}*. Treat each scope in *{RegisteredScopes* $\cap$ *RequestedScopes}* as matching any available scope associated with a resource found in the permission ticket.
3. For each *RequestedScopes(resource)* set, determine all operative policy conditions, and claims and other relevant information serving as input to them, and evaluate its authorization status.
4. For each scope in *RequestedScopes(resource)* that passes the evaluation, add it to a set called *CandidateGrantedScopes(resource)*.

Note: Claims and other information gathered during one authorization process may become out of date in terms of their relevance for future authorization processes. The authorization server is responsible for managing such relevance wherever information associated with a PCT, or other persistently stored information, is used as input to authorization, including policy conditions themselves.

Note: Since the authorization server's policy expression and evaluation capabilities are outside the scope of this specification, any one implementation might take a simple or arbitrarily complex form, with varying abilities to combine or perform calculations over claims and their values. For example, logical operations such as accepting "either claim value A or claim value B" as correct are possible to implement.

In the authorization results phase, the authorization server examines each *CandidateGrantedScopes(resource)* set to determine whether to issue an RPT and what permissions should be associated with it. If all *RequestedScopes(resource)* sets can be granted, then the authorization server subsequently responds with a success code and issues an RPT containing *CandidateGrantedScopes* for each resource.

Otherwise, the authorization server subsequently issues either an RPT containing *CandidateGrantedScopes* for each resource, or one of the error codes, as appropriate. The reason for the two options is that granting only partial scopes might not be useful for the client's and requesting party's purposes in seeking authorization for access. The choice of error depends on policy conditions and the authorization server's implementation choices. The conditions for the `need_info`, `request_denied`, and `request_submitted` error codes are dependent on authorization assessment and thus these codes might be more likely than the others to be issued subsequent to such a calculation.

The following example illustrates authorization assessment and partial results.

- The resource server has three of the resource owner's resources of interest to the client and requesting party, `photo1` and `photo2` with scopes `view`, `resize`, `print`, and `download`, and `album` with scopes `view`, `edit`, and `download`. It considers `photo1` and `photo2` to be logically "inside" `album`.
- Though the exact contents of RPTs, permissions, and permission requests are opaque to the client, the resource server has documented its API, available scopes, and permission requesting practices. For example, if the client requests an album resource, it expects that the resource server will request a permission for the album with a scope that approximates the attempted client operation, but will also request permissions for all the photos "inside" the album, with `view` scope only.
- The client has a pre-registered scope of `download` with the authorization server. This enables the client later to request this scope dynamically on behalf of its requesting party from the token endpoint. The authorization server assembles the set *RegisteredScopes* with contents of scope `download`.
- The client requests the album resource in an attempt to edit it, so the resource server obtains a permission ticket with three permissions in it: for `album` with a scope of `edit`, and for `photo1` and `photo2`, each with a scope of `view`. The authorization server assembles the following sets: *TicketScopes*(`album`) containing `edit`, *TicketScopes*(`photo1`) containing `view`, and *TicketScopes*(`photo2`) containing `view`.
- While asking for an RPT at the token endpoint, the client requests `download` scope on the requesting party's behalf. The authorization server determines the contents of the following sets: *RequestedScopes*(`album`) containing `edit` and `download`, *RequestedScopes*(`photo1`) containing `view` and `download`, and *RequestedScopes*(`photo2`) containing `view` and `download`.
- The resource owner has set policy conditions that allow access by this particular requesting party only to `photo1` and only for `view` scope.
- Based on the authorization server's authorization assessment calculation, it determines the contents of the following sets: *CandidateGrantedScopes*(`album`) containing no scopes, *CandidateGrantedScopes*(`photo1`) containing `view`, and *CandidateGrantedScopes*(`photo2`) containing no scopes. This adds up to less than in the corresponding *RequestedScopes* sets. The authorization server therefore has a choice whether to issue an RPT (in this case, containing a permission for `photo1` with `view` scope) or an error (say, `request_denied`, or `request_submitted` if has a way to notify the resource owner about the album editing resource request and seek an added policy covering it).

See Section 5.6 for a discussion of authorization implementation threats.

3.3.5 Authorization Server Response to Client on Authorization Success

If the authorization server's assessment process results in issuance of permissions, it issues the RPT with which it has associated the permissions by using the successful response form defined in Section 5.1 of [RFC6749].

The authorization server MAY return a refresh token. See Section 3.6 for more information about refreshing an RPT.

The authorization server MAY add the following parameters to its response:

pct OPTIONAL. A correlation handle representing claims and other information collected during this authorization process, which the client is able to present later in order to optimize future authorization processes on behalf of a requesting party. The PCT MUST be unguessable by an attacker. The PCT MUST NOT disclose claims from the requesting party directly to possessors of the PCT. Instead, such claims SHOULD be associated by reference to the PCT or expressed in an encrypted format that can be decrypted only by the authorization server that issued the PCT. See Section 3.3.2 for more information about the end-user requesting party interaction option. See Section 5.2 for additional PCT security considerations.

upgraded OPTIONAL. Boolean value. If the client submits an RPT in the request and the authorization server includes the permissions of the RPT from the request as part of the newly issued RPT, then it MUST set this value to `true`. If

it sets the value to `false` or the value is absent, the client MUST act as if the newly issued RPT does not include the permissions associated with the RPT from the request. (See Section 3.3.5.1.)

The authorization server MAY include any of the parameters defined in Section 5.1 of [RFC6749] on its response, except that it SHOULD NOT include the `scope` parameter. This is because for an RPT's permissions, each scope is associated with a specific resource, even though this association is opaque to the client. Note: The outcome of authorization assessment may result in expiration periods for RPTs, permissions, and refresh tokens that can affect the client's later requests for refreshing the RPT.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
...

{
   "access_token":"sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv",
   "token_type":"Bearer"
}
```

Example with a PCT in the response:

```
HTTP/1.1 200 OK
Content-Type: application/json
...

{
   "access_token":"sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv",
   "token_type":"Bearer",
   "pct":"c2F2ZWRjb25zZW50"
}
```

3.3.5.1 Authorization Server Upgrades RPT

The authorization server MAY implement RPT upgrading. The authorization server SHOULD document its practices regarding RPT upgrades and to act consistently with respect to RPT upgrades so as to enable clients to manage received RPTs efficiently.

If the authorization server has implemented RPT upgrading, the client has submitted an RPT in its request, and the result is success, the authorization server adds the permissions from the client's previous RPT to the RPT it is about to issue, setting the value of `upgraded` in its response containing the upgraded RPT to `true`.

If the authorization server is upgrading an RPT, and the RPT string is new rather than repeating the RPT provided by the client in the request, then the authorization server SHOULD revoke the existing RPT, if possible, and the client MUST discard its previous RPT. If the authorization server does not upgrade the RPT but issues a new RPT, the client MAY retain the existing RPT.

Example with `upgraded` in the response:

```
HTTP/1.1 200 OK
Content-Type: application/json
...

{
   "access_token":"sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv",
   "token_type":"Bearer",
   "upgraded":true
}
```

3.3.6 Authorization Server Response to Client on Authorization Failure

If the client's request to the token endpoint results in failure, the authorization server responds with an error, as defined in Section 5.2 of [RFC6749] and as follows.

invalid_grant If the provided permission ticket was not found at the authorization server, or the provided permission ticket has expired, or any other original reasons to use this error code are found as defined in [RFC6749], the authorization server responds with the HTTP 400 (Bad Request) status code.

invalid_scope At least one of the scopes included in the request does not match an available scope for any of the resources associated with requested permissions for the permission ticket provided by the client. The authorization server MAY also return this error when at least one of the scopes included in the request does not match a scope for which the client is pre-registered with the authorization server. The authorization server responds with the HTTP 400 (Bad Request) status code.

need_info The authorization server needs additional information in order for a request to succeed, for example, a provided claim token was invalid or expired, or had an incorrect format, or additional claims are needed to complete the authorization assessment. The authorization server responds with the HTTP 403 (Forbidden) status code. It MUST include a `ticket` parameter, and it MUST also include either the `required_claims` parameter or the `redirect_user` parameter, or both, as hints about the information it needs.

ticket REQUIRED. A permission ticket that allows the client to make a further request to the authorization server's token endpoint as part of this same authorization process, potentially immediately. The value MUST NOT be the same as the one the client used to make its request.

required_claims An array of objects that describe the required claims, with the following subparameters:

claim_token_format OPTIONAL. An array of strings specifying a set of acceptable formats for a claim token pushed by the client containing this claim, as defined in Section 3.3.1. Any one of the referenced formats would satisfy the authorization server's requirements. Each string MAY be a URI.

claim_type OPTIONAL. A string, indicating the expected interpretation of the provided claim value. The string MAY be a URI.

friendly_name OPTIONAL. A string that provides a human-readable form of the claim's name. This can be useful as a "display name" for use in user interfaces in cases where the actual name is complex or opaque, such as an OID or a UUID.

issuer OPTIONAL. An array of strings specifying a set of acceptable issuing authorities for the claim. Any one of the referenced authorities would satisfy the authorization server's requirements. Each string MAY be a URI.

name OPTIONAL. A string (which MAY be a URI) representing the name of the claim; the "key" in a key-value pair.

redirect_user The claims interaction endpoint URI to which to redirect the end-user requesting party at the authorization server to continue the process of interactive claims gathering, as defined in Section 3.3.2. For example, the authorization server could require the requesting party to log in to an account, or fill out a CAPTCHA to help prove humanness, or perform any number of other interactive tasks. If the requesting party is not an end-user, then no client action is possible on receiving the hint. If a static claims interaction endpoint was also provided in the authorization server's discovery document, then this value overrides the static value. Providing a value in this response might be appropriate, for example, if the URI needs to be customized per requesting party with a query parameter.

request_denied The client is not authorized to have these permissions. The authorization server responds with the HTTP 403 (Forbidden) status code.

request_submitted The authorization server requires intervention by the resource owner to determine whether the

client is authorized to have these permissions. The authorization server responds with the HTTP 403 (Forbidden) status code. It MUST include a `ticket` parameter and MAY include an `interval` parameter.

ticket REQUIRED. A permission ticket that allows the client to make one or more later polling requests to the token endpoint as part of this same authorization process, when the resource owner might have completed some approval (or denial) action. The value MUST NOT be the same as the one the client used to make its request.

interval OPTIONAL. The minimum amount of time in seconds that the client SHOULD wait between polling requests to the token endpoint. See Section 5.5 for security considerations in scenarios involving polling and consequences for permission ticket lifetimes.

Example when the permission ticket was not found or has expired:

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

{
   "error":"invalid_grant"
}
```

Example of a `need_info` response with hints about required claims:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Cache-Control: no-store
...

{
   "error":"need_info",
   "ticket":"ZXJyb3JfZGV0YWlscw==",
   "required_claims":[
      {
         "claim_token_format":[
            "http://openid.net/specs/openid-connect-core-1_0.html#IDToken"
         ],
         "claim_type":"urn:oid:0.9.2342.19200300.100.1.3",
         "friendly_name":"email",
         "issuer":[
            "https://example.com/idp"
         ],
         "name":"email23423453ou453"
      }
   ]
}
```

Example of a `need_info` response with a hint to redirect the requesting party to a claims interaction endpoint:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Cache-Control: no-store
...

{
   "error":"need_info",
   "ticket":"ZXJyb3JfZGV0YWlscw==",
   "redirect_user":"https://as.example.com/rqp_claims?id=2346576421"
}
```

Example when the client was not authorized to have the permissions:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Cache-Control: no-store
...

{
   "error":"request_denied"
}
```

Example when the authorization server requires resource owner intervention, including the optional `interval` parameter:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Cache-Control: no-store
...

{
   "error":"request_submitted",
   "ticket?:?ZXJyb3JfZGV0YWlscw==",
   "interval": 5
}
```

3.4 Client Requests Resource and Provides an RPT

The client requests the resource, now in possession of an RPT. The client uses [RFC6750] for a bearer token, and any other suitable presentation mechanism for an RPT of another access token type.

Example of a client request for the resource carrying an RPT:

```
GET /users/alice/album/photo.jpg HTTP/1.1
Authorization: Bearer sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv
Host: photoz.example.com
...
```

3.5 Resource Server Responds to Client's RPT-Accompanied Resource Request

The resource server responds to the client's RPT-accompanied resource request.

If the resource request fails, the resource server responds as if the request were unaccompanied by an access token, as defined in Section 3.2.

The resource server MUST NOT give access in the case of an invalid RPT or an RPT associated with insufficient authorization.

For an example of how the resource server can introspect the RPT and its permissions at the authorization server prior to responding to the client's request, see [UMAFedAuthz].

3.6 Authorization Server Refreshes RPT

As noted in Section 3.3.5, when issuing an RPT, the authorization server MAY also issue a refresh token.

Having previously received a refresh token from the authorization server, the client MAY use the refresh token grant as defined in [RFC6749] to attempt to refresh an expired RPT. If the client includes the `scope` parameter in its request, the authorization server MAY limit the scopes in the permissions associated with any resulting refreshed RPT to the scopes requested by the client.

The authorization server MUST NOT perform an authorization assessment calculation on receiving the client's request to

refresh an RPT.

3.7 Client Requests Token Revocation

If the authorization server presents a token revocation endpoint as defined in [RFC7009], the client MAY use the endpoint to request revocation of an RPT (access token), refresh token, or PCT previously issued to it on behalf of a requesting party. This specification defines the following token type hint value:

pct Helps the authorization server optimize lookup of a PCT for revocation.

4. Profiles and Extensions

An UMA profile restricts UMA's available options. An UMA extension defines how to use UMA's extensibility points. The two can be combined. Some reasons for creating profiles and extensions include:

- A profile restricting options in order to tighten security
- A profile/extension restricting options and adding messaging parameters for use with a specific industry API
- A profile that documents a specific URI, format, and interpretation for pushed claim tokens (see Section 3.3.1)
- An extension that defines additional metadata for the authorization server discovery document to define machine-readable usage details

The following actions are RECOMMENDED regarding the creation and use of profiles and extensions:

- The creator of a profile or extension related to UMA SHOULD assign it a uniquely identifying URI.
- The authorization server supporting a profile or extension related to UMA with such a URI SHOULD supply the identifying URI in its `uma_profiles_supported` metadata (see Section 2).

5. Security Considerations

This specification relies mainly on OAuth 2.0 security mechanisms as well as transport-level security. Thus, implementers are strongly advised to read [BCP195] and the security considerations in [RFC6749] (Section 10) and [RFC6750] (Section 5) along with the security considerations of any other OAuth token-defining specifications in use, along with the entire [RFC6819] specification, and apply the countermeasures described therein. As well, implementers should take into account the security considerations in all other normatively referenced specifications.

The following sections describe additional security considerations.

5.1 Cross-Site Request Forgery

Redirection used for gathering claims interactively from an end-user requesting party (described in Section 3.3.2) creates the potential for cross-site request forgery (CSRF). This may be the result of an open redirect if the authorization server does not force the client to pre-register its claims redirection endpoint, and server-side artifact tampering if the client does not avail itself of the `state` parameter.

A CSRF attack against the authorization server's claims interaction endpoint can result in an attacker obtaining authorization for access through a malicious client without involving or alerting the end-user requesting party. The authorization server MUST implement CSRF protection for its claims interaction endpoint and ensure that a malicious client cannot obtain authorization without the awareness and involvement of the requesting party.

If the client uses the interactive claims gathering feature, it MUST implement CSRF protection for its claims redirection URI. It SHOULD use the `state` parameter when redirecting the requesting party to the claims interaction endpoint. The value of the `state` parameter MUST be unguessable by an attacker. Once the authorization server redirects the requesting party back, with the required binding value contained in the `state` parameter, the client MUST check that the value of the `state` parameter received is equal to the value sent in the initial redirection request. Depending on the type of application, a client has several methods for storing and later verifying the value of the `state` parameter in between the initial redirect and the eventual resulting request to the claims redirection URI, including storage in a server-side session-bound variable, cryptographic derivation from a browser cookie, or secure application-level storage. The client MUST treat requests containing an invalid or unknown `state` parameter value as an error.

The `state` parameter SHOULD NOT include sensitive client or requesting party information in plain text, as it is transmitted through third-party components (the requesting party's user agent) and could be stored insecurely.

5.2 RPT and PCT Exposure

When a client redirects an end-user requesting party to the claims interaction endpoint, the client provides no a priori context to the authorization server about which user is appearing at the endpoint, other than implicitly through the permission ticket. Thus, a malicious client has the opportunity to switch end-users -- say, enabling malicious end-user Carlos to impersonate legitimate end-user Bob, who might be represented by a PCT already in that client's possession and might even have authorized the issuance of that PCT -- after the redirect completes and before it returns to the token endpoint to seek permissions.

To mitigate this threat, the authorization server, with the support of the resource owner, should consider the following strategies in combination.

- Require that the requesting party legitimately represent the wielder of the RPT on a legal or contractual level. This solution alone does not reduce the risk from a technical perspective.
- Gather claims interactively from an end-user requesting party that demonstrate that some sufficiently strong level of authentication was performed.
- Require claims to have a high degree of freshness in order for them to satify policy conditions.
- Tighten time-to-live strategies around RPTs and their associated permissions (see Section 6.1).

The client MUST only share the RPT (access token) with the resource server and authorization server, as explained in Section 10.3 of [RFC6749], and thus MUST keep it confidential from the requesting party. Because a malicious requesting party (the user of the client in the UMA grant) may have incentives to steal an RPT that the resource owner (the user of the client in other OAuth grants) does not, this security consideration takes on especial importance.

The PCT is similar to a refresh token in that it allows non-interactive issuance of access tokens. The authorization server and client MUST keep the PCT confidential in transit and storage, and MUST NOT share the PCT with any entity other than each other. The authorization server MUST maintain the binding between the PCT and the client to which it was issued.

Given that the PCT represents a set of requesting party claims, a client supplying a PCT in its RPT request MUST make a best effort to ensure that the requesting party using the client now is the same as the requesting party that was associated with the PCT when it was issued. Different clients will have different capabilities in this respect; for example, some applications are single-user and perform no local authentication, associating all PCTs with the "current user", while others might have more sophisticated authentication and user mapping capabilities.

If the authorization server has reason to believe that a PCT is compromised, for example, if the PCT has been supplied by a client that has "impossible geography" parameters, the authorization server should consider not using the claims based on that PCT in its authorization assessment.

5.3 Strengthening RPT Protection Using Proof of Possession

After the client's resource request with an RPT, assuming the client sent an RPT of the bearer style such as defined in [RFC6750], the resource server will have received from the client the entire secret portion of the token. This specification assumes only bearer-type tokens because they are the only type standardized as of this specification's publication. However, to strengthen protection for RPTs using a proof-of-possession approach, the resource server could receive an RPT that consists of only a cryptographically signed token identifier, and then to validate the signature, it could, for example, submit the token identifier to the token introspection endpoint to obtain the necessary key information. The details of this usage are outside the scope of this specification.

5.4 Credentials-Guessing

Permission tickets and PCTs are additional credentials that the authorization server MUST prevent attackers from guessing, as defined in Section 10.10 of [RFC6749].

5.5 Permission Ticket Management

Within the constraints of making permission ticket values unguessable, the authorization server MAY format the permission ticket however it chooses, for example, either as a random string that references data held on the server or by including data within the ticket itself.

Permission tickets MUST be single-use. This prevents susceptibility to a session fixation attack.

The authorization server MUST invalidate a permission ticket when the client presents the permission ticket to either the token endpoint or the claims interaction endpoint, or when the permission ticket expires, whichever occurs first.

The client SHOULD check that the value of the `ticket` parameter it receives back from the authorization server in each response and each redirect of the requesting party back to it differs from the one it sent to the server in the initial request or redirect.

If the authorization server has reason to believe that a permission ticket is compromised, for example, because it has seen the permission ticket before and it believes the first appearance was from a legitimate client and the second appearance is from an attacker, it should consider invalidating any access tokens based on this evidence.

Given that scenarios involving the `request_submitted` error code are likely to involve polling intervals, the permission

ticket needs to last long enough to give the client a chance to attempt a polling request, which then needs to figure into other permission ticket security considerations.

5.6 Naive Implementations of Default-Deny Authorization

While a reasonable approach for most scenarios is to implement the classic stance of default-deny ("everything that is not expressly allowed is forbidden"), corner cases can inadvertently result in default-permit behavior. For example, it is insufficient to create default "empty" policy conditions stating "no claims are needed", and then accept an empty set of supplied claims as sufficient for access during authorization assessment.

5.7 Requirements for Pre-Established Trust Regarding Claim Tokens

When a client makes an RPT request, it has the opportunity to push a claim token to attempt to satisfy policy conditions (see Section 3.3.1).

Claim tokens of any format typically contain audience restrictions, and an authorization server would not typically be in the primary audience for a claim token held or generated by a client. It is RECOMMENDED to document how the client, authorization server, requesting party, and any additional ecosystem entities and parties will establish a trust relationship and communicate any required keying material in a claim token profile, as described in Section 4. Authorization servers are RECOMMENDED not to accept claim tokens pushed by untrusted clients and not to ignore audience restrictions found in claim tokens pushed by clients.

A malicious client could push a claim token to the authorization server (revealing the claims therein; see Section 6.2) to seek resource access on its own behalf prior to any opportunity for an end-user requesting party to authorize claims collection. It is RECOMMENDED either for trust relationships established by the ecosystem parties to include prior requesting party authorization as required, or for end-user requesting party authorization to be gathered interactively prior to claims pushing, as described in Section 3.3.2.

Some deployments could have exceptional circumstances allowing the authorization server to validate claim tokens. For example, if the authorization server itself is also the identity provider for the requesting party, then it would be able to validate any ID token that the client pushes as a claim token and also validate the client to which it was issued.

5.8 Profiles and Trust Establishment

Parties that are operating and using UMA software entities may need to establish agreements about the parties' rights and responsibilities on a legal or contractual level, along with common interpretations of UMA constructs for consistent and expected software behavior. These agreements can be used to improve the parties' respective security postures. Written profiles are a key mechanism for conveying and enforcing these agreements. Section 4 discusses profiling. See [UMA-legal] to learn about frameworks and tools to assist in the legal and contractual elements of deploying UMA-enabled services.

6. Privacy Considerations

UMA has the following privacy considerations.

6.1 Policy Condition Setting, Time-to-Live Management, and Removal of Authorization Grants

The setting of policy conditions, the resource owner-authorization server interface, and the resource owner-resource server interface are outside the scope of this specification. (For an example of how a secure and authorized resource owner context can be established between the resource server and authorization server, see [UMAFedAuthz].)

A variety of flows and user interfaces for policy condition setting involving user agents for both of these servers are possible, each with different privacy consequences for end-user resource owners. As well, various authorization, security, and time-to-live strategies could be applied on a per-resource owner basis or a per-authorization server basis, as the entities see fit. Validity periods of RPTs, refresh tokens, permissions, caching periods for responses, and even OAuth client credentials are all subject to management. Different time-to-live strategies might be suitable for different resources and scopes.

In order to account for modifications of policy conditions that result in the withdrawal of authorization grants (for example, fewer scopes, fewer resources, or resources available for a shorter time) in as timely a fashion as possible, the authorization server should align its strategies for management of these factors with resource owner needs and actions rather than those of clients and requesting parties. For example, the authorization server may want to invalidate a client's RPT and refresh token as soon as a resource owner changes policy conditions in such a way as to deny the client and its requesting party future access to a full set of previously held permissions.

6.2 Requesting Party Information at the Authorization Server

Claims are likely to contain personal, personally identifiable, and sensitive information, particularly in the case of requesting parties who are end-users.

If the authorization server supports persisting claims for any length of time (for example, to support issuance of PCTs), then it SHOULD provide a secure and privacy-protected means of storing claim data. It is also RECOMMENDED for the authorization server to use an interactive claims-gathering flow to ask an end-user requesting party for authorization to collect any claims subsequently and to persist their claims (for example, before issuing a PCT), if no prior requesting party authorization has been established among the ecosystem parties (see Section 5.7).

6.3 Resource Owner Information at the Resource Server

Since the client's initial request for a protected resource is made in an unauthorized and unauthenticated context, such requests are by definition open to all users. The response to that request includes the authorization server's location to enable the client to request an access token and present claims. If it is known out of band that authorization server is owned and controlled by a single user, or visiting the authorization server contains other identifying information, then an unauthenticated and unauthorized client would be able to tell which resource owner is associated with a given resource. Other information about the resource owner, such as organizational affiliation or group membership, may be gained from this transaction as well.

6.4 Profiles and Trust Establishment

Parties that are operating and using UMA software entities may need to establish agreements about mutual rights, responsibilities, and common interpretations of UMA constructs for consistent and expected software behavior. These agreements can be used to improve the parties' respective privacy postures. See Section 5.8 for more information. Additional considerations related to Privacy by Design concepts are discussed in [UMA-PbD].

7. IANA Considerations

This document makes the following requests of IANA.

7.1 Well-Known URI Registration

This specification registers the well-known URI defined in Section 2, as required by Section 5.1 of [RFC5785].

7.1.1 Registry Contents

- URI suffix: `uma2-configuration`
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 2 in this document

7.2 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].

7.2.1 Registry Contents

- Metadata name: `claims_interaction_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: `uma_profiles_supported`
- Metadata description: profile/extension feature metadata
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 2 in this document

7.3 OAuth 2.0 Dynamic Client Registration Metadata Registry

This specification registers OAuth 2.0 dynamic client registration metadata defined in Section 2, as required by Section 4.1 of [RFC7591].

7.3.1 Registry Contents

- Metadata name: `claims_redirect_uris`
- Metadata description: claims redirection endpoints
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 2 in this document

7.4 OAuth 2.0 Extension Grant Parameters Registration

This specification registers the parameters defined in Section 3.3.1, as required by Section 11.2 of [RFC6749].

7.4.1 Registry Contents

- Parameter name: `claim_token`

- Parameter usage location: client request, token endpoint
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.1 in this document

- Parameter name: `pct`
- Parameter usage location: client request, token endpoint
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.1 in this document

- Parameter name: `pct`
- Parameter usage location: authorization server response, token endpoint
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.5 in this document

- Parameter name: `rpt`
- Parameter usage location: client request, token endpoint
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.1 in this document

- Parameter name: `ticket`
- Parameter usage location: client request, token endpoint
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.1 in this document

- Parameter name: `upgraded`
- Parameter usage location: authorization server response, token endpoint
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.5 in this document

7.5 OAuth 2.0 Extensions Error Registration

This specification registers the errors defined in Section 3.3.6, as required by Section 11.4 of [RFC6749].

7.5.1 Registry Contents

- Error name: `need_info` (and its subsidiary parameters)
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.6 in this document
- Error usage location: authorization server response, token endpoint

- Error name: `request_denied`
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.6 in this document
- Error usage location: authorization server response, token endpoint

- Error name: `request_submitted` (and its subsidiary parameters)
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.3.6 in this document
- Error usage location: authorization server response, token endpoint

7.6 OAuth Token Type Hints Registration

This specification registers the errors defined in Section 3.7, as required by Section 4.1.2 of [RFC7009].

7.6.1 Registry Contents

- Hint value: `pct`
- Change controller: Kantara Initiative User-Managed Access Work Group - staff@kantarainitiative.org
- Specification document: Section 3.7 in this document

8. 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].

9. References

9.1 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>.

[OIDCDynClientReg] Sakimura, N., "OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-registration-1_0.html>.

[UMAFedAuthz] Maler, E., "Federated Authorization for User-Managed Access (UMA) 2.0", January 2018, <https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-federated-authz-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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[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>.

[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>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[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>.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/info/rfc7009>.

9.2 Informative References

[UMA-PbD] Maler, E., "Privacy by Design Implications of UMA", 2018, <https://kantarainitiative.org/confluence/display/uma/Privacy+by+Design+Implications+of+UMA>.

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

[UMA-legal] Maler, E., "UMA Legal", 2017, <http://kantarainitiative.org/confluence/display/uma/UMA+Legal>.

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