OPEN AUTHENTICATION TECHNICAL COMMITTEE

Online Multimedia Authorization Protocol

An Industry Standard for Authorized Access to Internet Multimedia Resources

Joel Huff, Adobe Systems; David Schlacht, DirecTV; Anthony Nadalin, Microsoft; John Simmons, Microsoft; Peter Rosenberg, NBC Universal; Paul Madsen, Ping Identity; Tim Ace, Synacor; Cyril Rickelton-Abdi, Turner; Bill Boyer, Viacom.

8/22/2012

Version 1.0

A multimedia authorization protocol based on OAuth 2.0, using JSON Web Tokens.

Contents

Figures

Tables

ONLINE MULTIMEDIA AUTHORIZATION PROTOCOL

Joel Huff, Adobe Systems; David Schlacht, DirecTV; Anthony Nadalin, Microsoft; John Simmons, Microsoft; Peter Rosenberg, NBC Universal; Paul Madsen, Ping Identity; Tim Ace, Synacor; Cyril Rickelton-Abdi, Turner; Bill Boyer, Viacom.

1 INTRODUCTION

Multimedia content publishers and distributors are expanding their reach beyond traditional outlets to include a broad range of Internet-connected devices, enabling greater consumer access, choice and convenience. Extending subscription and other business models to provide consumer access to media content over the Internet requires new methods of securely authorizing End-users, their devices and client software.

For content owners and distributors, these methods must provide the means to prevent unauthorized access to content while providing a common architecture and framework for interoperability, minimizing the operational overhead and cost of implementing divergent protocols.

For the consumer, these methods must provide a simple and consistent experience, avoiding unfamiliar, intrusive or frequent logins, which could discourage usage and broad adoption. They should enable sites and applications to provide customized search-results, navigation and personalization while ensuring that the consumer's identity and privacy are adequately protected.

For technology vendors and 3rd-party service providers, these methods must provide interoperability, enabling them to develop value-added services and commercial solutions that can be leveraged across the entire ecosystem.

1.1 SCOPE

This specification addresses Authorization — the process of granting or denying access to network resources, such as protected multimedia assets. It defines the architecture, protocols and data formats needed to build and deploy interoperable systems that authorize access to protected media resources on any Internet-connected device. End-user Authentication, which validates the consumer's identity, is distinct from Authorization and out of scope. Various Authentication methods, such as HTML form-based login or SAML 2.0, may be used as needed in conjunction with this specification (see section 5.3 for additional information).

1.2 UNDERLYING PROTOCOL AND DATA FORMATS

This specification is compliant with the OAuth 2.0 protocol. OAuth 2.0 provides support for both Web and Native Applications, is compatible with REST frameworks, has strong developer support, and has broad and growing industry adoption.

The OAuth 2.0 specification uses JSON data structures. JSON (JavaScript Object Notation) is a lightweight, text-based data interchange format for the portable representation of structured data. This specification also uses additional JSON-based specifications for web tokens, encryption and digital signatures.

1.3 CONVENTIONS

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

- “MUST”, “REQUIRED” and “SHALL” mean that the definition is an absolute requirement of the specification.
- “MUST NOT” and “SHALL NOT” mean that the definition is an absolute prohibition of the specification.
- “SHOULD” and “RECOMMENDED” mean that there may be valid reasons to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- “SHOULD NOT” and “NOT RECOMMENDED” mean that there may be valid reasons when the particular behavior is acceptable, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- “MAY” and “OPTIONAL” mean that the item is truly optional.

1.4 TERMINOLOGY AND ABBREVIATIONS

1.4.1 TERMINOLOGY

Access Conditions	A character string consisting of name-value pairs that describe a set of Protected Resources and the circumstances or context in which access to them might occur. Clients specify their desired Access Conditions in Authorization Requests, and Authorization Servers specify granted Access Conditions in Authorization Responses. Access Conditions are passed to the Authorization Server in the "scope" parameter of the Authorization Request. They are encoded in the JWT Claims Set of the Access Token by the Authorization Server. They may also be included in the "scope" field of the Access Token Response, if the access granted by the Authorization Server differs from the access requested by the Client.
Access Conditions Subset	When Access Conditions consist of the logical union of multiple space-delimited conditions, each of those conditions is called an Access Conditions Subset, or simply a Subset.
Access Token	A character string enabling access to Protected Resources under specified Access Conditions. An Access Token is provided to a Client by an Authorization Server in the Access Token Response. It is then used by the Client to access Protected Resources under the protection of a Resource Server.
Access Token Request	Request made by the Client to the Authorization Server to exchange an Authorization Code for Access and Refresh Tokens.
Access Token Response	The response from the Authorization Server to the Client's Access Token Request.
Authentication	Any method used to identify an End-user to an Authorization Provider. Authentication may also refer to a method used to identify a Client to the Authorization Server.
Authentication Session	An End-user Authentication state extending over multiple OMAP requests, maintained by the Authorization Server using any of a variety of session management techniques, such as the use of HTTP cookies.
Authorization	The process that determines whether an End-user is permitted access to specific Protected Resources.

Authorization Code	An Authorization Code is an OAuth 2.0 Authorization Grant type used in OMAP to obtain both Access Tokens and Refresh Tokens. ("1.4.1 Authorization Code", [OAUTH]).
Authorization Endpoint	The Authorization Endpoint is the endpoint that responds to Authorization Requests sent by the User-agent on behalf of the End-user. See also Token Endpoint and Status Endpoint.
Authorization Grant	Intermediate credentials representing the Resource Owner's authorization to access its Protected Resources. An Authorization Grant is used by the Client to obtain an Access Token ("1.4 Authorization Grant", [OAUTH]). The type of Authorization Grant used in this specification is an Authorization Code.
Authorization Provider (AzP)	An entity that authorizes End-users to access Protected Resources.
Authorization Request	The HTTP request directed to an Authorization Server to request access to Protected Resources under specified Access Conditions.
Authorization Response	The response from the Authorization Server to the Authorization Request. If the Resource Owner grants the access request, the Authorization Server delivers an Authorization Code to the Client.
Authorization Server (AS)	The server that issues Access Tokens to the Client after receiving authorization from the Resource Owner.
Authorization Status Request/Response	The Authorization Status Request and Response provide a lightweight means for a Client to determine the End-user's permissions to access Protected Resources under specified Access Conditions without actually requesting an Access Token.
Client	The Client is the application making Protected Resource requests on behalf of an End-user with the authorization of the Resource Owner.
Client Identifier	In OAuth 2.0, a unique string representing the registration information provided by the Client.
Client Registration	A process that identifies the Client to an Authorization Provider and provides the Client with a Client Identifier and optionally a client secret (see [RFC2617]).
Content Provider (CP)	In OMAP, the Content Provider makes Protected Resources available to the End-user by means of an authorized Client.

Device Binding ID (DBID)	A device identifier, optionally provided by a Client in the Authorization Request, which can be encoded by the Authorization Server into an Access Token, effectively "binding" the Access Token to a device or a domain of devices.
Device Binding Type (DBT)	A string designating the method used to generate the Device Binding ID, optionally provided by a Client in the Authorization Request. The Authorization Server can encode this with the DBID in an Access Token, effectively "binding" the Access Token to a device or domain of devices and the type of method used to create the DBID.
Device Management ID (DMID)	A device identifier used by an Authorization Provider to securely identify, count, limit and manage devices associated with an End-user account.
End-user	Individual using a Client application for the purpose of accessing Protected Resources with the permission of an Authorization Provider.
Identity Provider (IdP)	An MVPD, OVD, or 3rd-party vendor responsible for providing End-user Authentication services.
JavaScript Object Notation (JSON)	A lightweight text-based open standard designed for human-readable data exchange.
JSON Web Encryption (JWE)	JWE is a means of representing encrypted content using JSON data structures.
JSON Web Signature (JWS)	JWS is a means of representing signed content using JSON data structures.
JSON Web Token (JWT)	JWT is a means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that may be digitally signed using a JSON Web Signature (JWS) and/or encrypted using JSON Web Encryption (JWE).
JWT Claims Set	A string representing a JSON object containing the claims conveyed by a JSON Web Token (JWT).
Multichannel Video Programming Distributor (MVPD)	An FCC term that includes cable, satellite and telco providers. See also Authorization Provider and OVD.
Native Application	A Native Application is a Client installed and executed on an End-user's device.

OAuth 2.0 (OAuth)	An open standard for authorization enabling the sharing of restricted resources by handing out tokens instead of credentials.
Obligation Data	Information provided by the Authorization Server in either the Access Token Response or the Refresh Token Response indicating to the Client the conditional obligations associated with accessing the Protected Resources.
Online Multimedia Authorization Protocol (OMAP)	OAuth 2.0-based protocol enabling Authorization Providers to convey an End-user's permissions to access Protected Resources.
Online Video Distributor (OVD)	FCC term for non-MVPD providers of internet video services. See also Multichannel Video Programming Distributor and Authorization Provider.
Protected Resource	Any content, media or data that requires the End-user to be granted Authorization in order to obtain access.
Refresh Token	Refresh Tokens are credentials issued to a Client by an Authorization Server to be used to obtain a new Access Token when the current Access Token becomes invalid, expires, or is determined to have the wrong Access Conditions.
Remediation Data	Information provided by the Authorization Server in the Authorization Response, the Access Token Response or the Refresh Token Response when access is denied or only partially granted. The Client MAY use Remediation Data to enable the End-user to remedy the problem. Remediation Data may also be provided by the Authorization Server in the Authorization Status Response when access would be denied or only partially granted were the Client to make an Authorization Request.
Resource Owner (RO)	In OAuth 2.0, the Resource Owner is the entity capable of granting access to a Protected Resource. In OMAP the Resource Owner is an MVPD or an OVD.
Resource Server (RS)	In OAuth 2.0, a Resource Server provides access to Protected Resources by accepting and responding to protected resource requests from a Client. In OMAP, the Resource Server role is played by the Service Provider on behalf of the Content Provider.
Resource Usage Monitor (RUM)	An online service that collects and aggregates data about resource usage, enabling real-time analysis and decisions based on End-user usage patterns. Detection and prevention of fraud and abuse are possible applications of such a service.

SAML Assertions	SAML assertions carry statements about a principal that an asserting party claims to be true. There are three types of SAML statements for encoding user identity and privileges: SAML authentication statements, SAML authorization statements and SAML attributes.
Security Assertion Markup Language (SAML)	SAML is a standard for exchanging authentication and authorization data between Identity Providers (IdP) and service providers (SP).
Service Provider (SP)	A Service Provider provides an End-user access to a Content Provider's Protected Resources, with the permission of an Authorization Provider.
Service Provider ID (SPID)	A string or URI designating the Service Provider for the requested Protected Resource.
Single sign-on (SSO)	The access control feature that allows an End-user to log in just once to gain access to multiple related, but independent software systems, such as Service Providers.
Status Endpoint	The Status Endpoint responds to Authorization Status Requests from either the User-agent or the Client on behalf of the End-user. See Authorization Endpoint and Token Endpoint.
Subset	See Access Condition Subset
Time to live (TTL)	Time to live is a mechanism limiting the lifetime of data on a network. In OMAP, it refers to the lifetime of the Access Token.
Token Endpoint	The Token Endpoint is used by the Client to obtain Access and Refresh Tokens, by presenting its Authorization Grant or Refresh Token. See Authorization Endpoint and Status Endpoint.
Uniform Resource Identifier (URI)	A URI is a string of characters used to identify a name or a resource. A URI can be a locator (URL) or a name (URN), used to enable interaction with a representation of the resource on a network. See URL and URN.
Uniform Resource Locator (URL)	A URL is a string of characters used to locate a resource. See URI and URN.
Uniform Resource Name (URN)	A URN is a string of characters giving the identity of a resource. See URI and URL.
User-agent	In OMAP, this is the client software used by the End-user to access HTTP servers. In the Authorization Request/Response flow, the User-agent serves as an intermediary between the Client and the Authorization Server.

User-agent-based Application	In OAuth 2.0, a User-agent-based application is "... a Client in which the client code is downloaded from a web server and executes within a user-agent on the resource owner's device."
Web Application	A Web Application is a Client in which the Client code runs on a web server.

1.4.2 ABBREVIATIONS AND ACRONYMS

AS	Authorization Server
AuthN	Authentication
AuthZ	Authorization
AzP	Authorization Provider
CP	Content Provider
DBID	Device Binding ID
DBT	Device Binding Type
DMID	Device Management ID
IdP	Identity Provider
JSON	JavaScript Object Notation
JWE	JSON Web Encryption
JWS	JSON Web Signature
JWT	JSON Web Token
MVPD	Multichannel Video Programming Distributor
OMAP	Online Multimedia Authorization Protocol
OVD	Online Video Distributor
RO	Resource Owner
RS	Resource Server
RUM	Resource Usage Monitor
SAML	Security Assertion Markup Language
SP	Service Provider
SPID	Service Provider ID
SSO	Single sign-on
TTL	Time to live
URI	Uniform Resource Identifier (URI)

URL	Uniform Resource Locator (URL)
URN	Uniform Resource Name (URN)

1.5 REFERENCES

1.5.1 NORMATIVE REFERENCES

[HTML] Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification," World Wide Web Consortium Recommendation REC-html401-19991224, December 1999 (HTML).

[JWA] "JSON Web Algorithms (JWA)", http://datatracker.ietf.org/doc/draft-ietf-jose-json-web-algorithms/

[JWE] "JSON Web Encryption (JWE)", http://datatracker.ietf.org/doc/draft-ietf-jose-json-web-encryption/

[JWS] "JSON Web Signature (JWS)", http://datatracker.ietf.org/doc/draft-ietf-jose-json-web-signature/

[JWT] "JSON Web Token (JWT)", http://tools.ietf.org/html/draft-jones-json-web-token

[OAUTH] "The OAuth 2.0 Authorization Protocol", standards track Internet draft, Internet Engineering Task Force (IETF), http://tools.ietf.org/html/draft-ietf-oauth-v2

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2141] "URN Syntax", May 1997, http://www.ietf.org/rfc/rfc2141.txt

1.5.2 INFORMATIONAL REFERENCES

[OAUTHSML] "SAML 2.0 Bearer Assertion Profiles for OAuth 2.0", http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer

[OAUTHDV] "OAuth 2.0 Device Profile", standards track Internet draft, Internet Engineering Task Force (IETF), http://tools.ietf.org/html/draft-recordon-oauth-v2-device

[OICREG] "OpenID Connect Dynamic Client Registration 1.0 ", Jan 2012, http://openid.net/specs/openid-connect-registration-1_0.html

[RFC2616] "Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999, http://tools.ietf.org/html/rfc2616

[RFC2617] "HTTP Authentication: Basic and Digest Access Authentication", J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, June 1999, http://tools.ietf.org/html/rfc2617

[SAML] "Assertions and Protocols for the OASIS Security Assertion markup Language (SAML) V2.0", OASIS Standard, March 2005, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[SAMLGLOSS] "Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0" OASIS SSTC, J. Hodges et al, March 2005, http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf

[SWD] "Simple Web Discovery (SWD)", http://wiki.tools.ietf.org/html/draft-jones-simple-web-discovery-00

2 HIGH LEVEL DESIGN

This section provides an overview of OMAP - the actors and their roles in the protocol, the nominal Web Application and Native Application protocol flows, and the expression of Access Conditions in the OAuth 2.0 “scope” parameter and in Access Tokens.

2.1 ECOSYSTEM PARTICIPANTS (INFORMATIONAL)

One of the principal use cases addressed by OMAP is shown in Figure 1. In this example (1) a consumer requests access to video content on a television programmer’s website, (2) authenticates as a paid subscriber of their MVPD which (3) authorizes the programmer to grant the subscriber access to (4) view the website’s content.

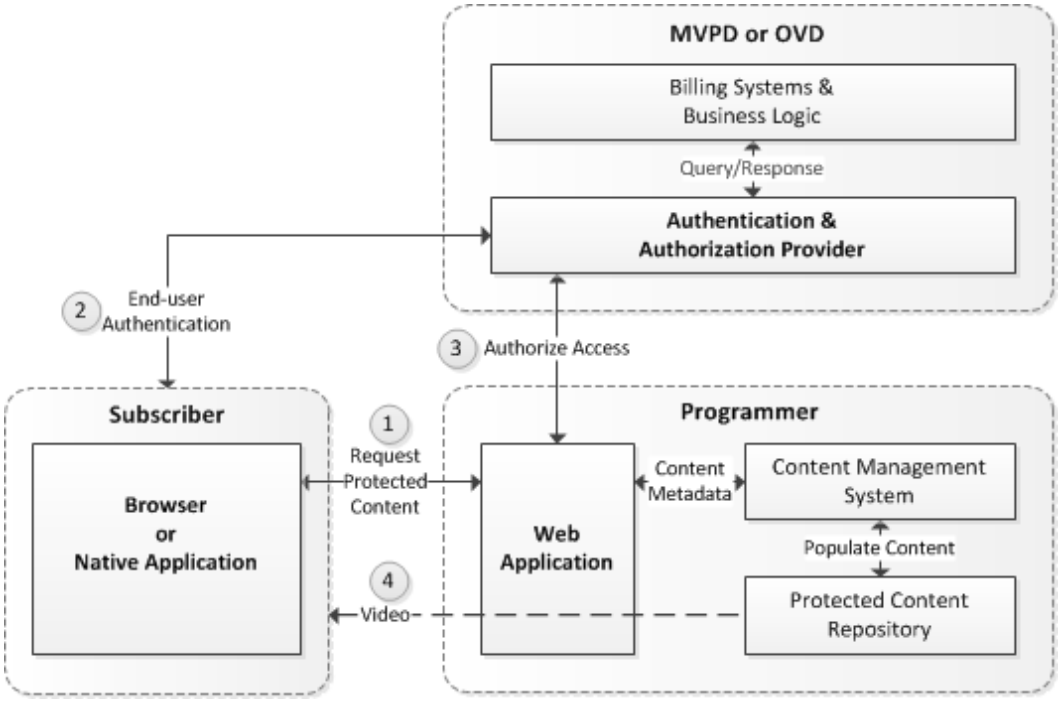

Figure 1 – A Principal OMAP Use Case

OMAP uses the OAuth 2.0 protocol to support this and similar use cases. Table 1 and Figure 2 below summarize the mapping between OAuth 2.0 roles and the actors in an OMAP ecosystem.

Table 1 – OMAP Roles and Actors

OMAP Roles	Actors	Example
End-user	**Consumer**	Paid subscriber of an MVPD or OVD
Resource Owner	**Authorization Provider**	MVPD with subscriber rights database

OMAP Roles	Actors	Example
Authorization Server	**Authorization Provider**	Authorization Server hosted by an MVPD or an OVD
Resource Server	**Content Provider**, Authorization Provider	Resource Server hosted by a Service Provider, a programmer, a syndication partner, an MVPD or an OVD
Client	**Content Provider**, Authorization Provider	Web Application provided by a Content, Service or Authorization Provider, or a Native Application
User-Agent	**Browser**, Native Application	Device or personal computer application interface between End-user and Client

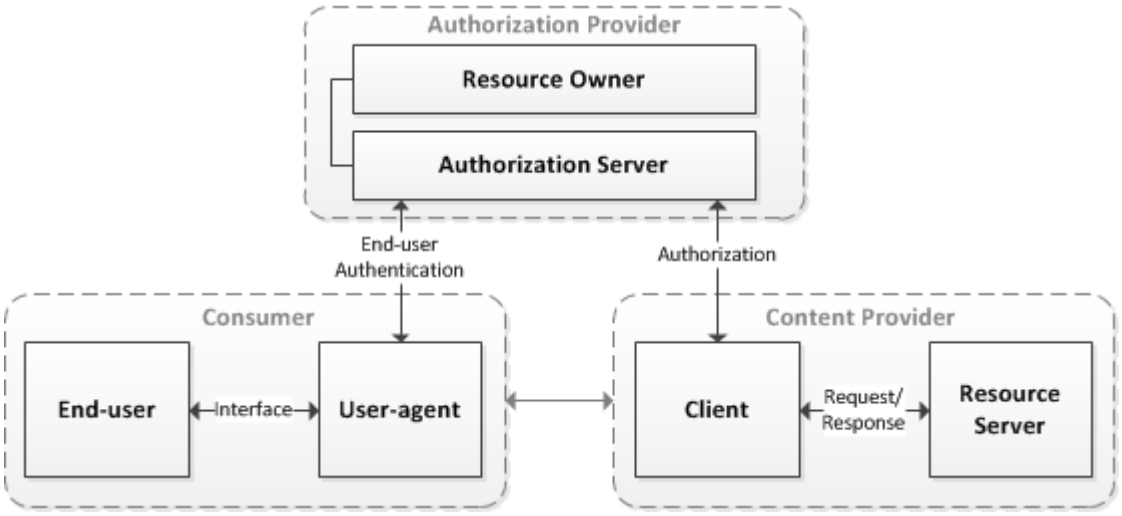

Figure 2 – OMAP Roles and Actors

In OAuth 2.0 the role of Resource Owner (RO) is described as the "entity capable of granting access to a protected resource". In OMAP the Resource Owner is the Authorization Provider (AzP). The End-user is entitled to access Protected Resources by virtue of his/her relationship with the AzP. The Authorization Server (AS) is described as "the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization." It is important to note that in the OMAP ecosystem the RO and AS are commonly both hosted by the AzP. In such instances where the RO and AS are collocated, authentication of the RO (as distinct from End-user Authentication) may be unnecessary.

The Client is described in OAuth 2.0 as "an application making protected resource requests on behalf of the resource owner and with its authorization". As previously noted, the RO in OMAP refers to the Authorization Provider, so in OMAP it is more accurate to state that "the Client is

an application making protected resource requests on behalf of the End-User and with the authorization of the Resource Owner".

In OMAP, as in OAuth 2.0, the Resource Server (RS) is "the server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens".

2.2 WEB APPLICATION PROTOCOL FLOW

The high level architecture consists of interactions between Content Providers, Authorization Providers, Clients and End-users. In the case of a Web Application, the Client consists of the Web Application on the Content Provider site interacting with the User-agent of the End-user's device.

Clients obtain tokens from Authorization Servers that express the End-user's permissions to access Protected Resources. These tokens are presented to the Content Provider's Resource Server with requests to access Protected Resources. By examining the permissions expressed in these tokens, the Resource Server can decide whether or not to provide access to the requested Protected Resource.

Figure 3 summarizes the OMAP OAuth 2.0 protocol flow for Web Applications. The flow is nearly identical to the OAuth 2.0 Authorization Code protocol flow (see "Protocol Flow", [OAUTH]), except in step (B). In OMAP, the entity that can authorize access to Protected Resources is not the End-user, but an entity owning the rights to authorize distribution of the Protected Resources, such as an MVPD or OVD. It is this entity that acts as the Authorization Provider and may grant access to Protected Resources by verifying its relationship to the End-user, who may be entitled to such access by virtue of a paid subscription or other considerations.

By using the OAuth 2.0 Authorization Code flow OMAP obtains its security advantage – namely, the ability to authenticate the Client and deliver the access token directly to the Client without exposing it to the End-user's User-agent.

The following steps (A) through (G) describe the OMAP web application flow shown in Figure 3. The text comes from the OAuth 2.0 Authorization Code Protocol Flow [OAUTH]. The underlined text in steps (A) and (B) are the only departure from the OAuth 2.0 text.

(A) **Authorization Request.** The client initiates the flow by directing the End-user's User-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

> *The difference here from the [OAUTH]-defined Authorization Code protocol flow is that it is not the User-agent of the Resource Owner, but of an End-user (e.g., an MVPD subscriber) that is involved.*

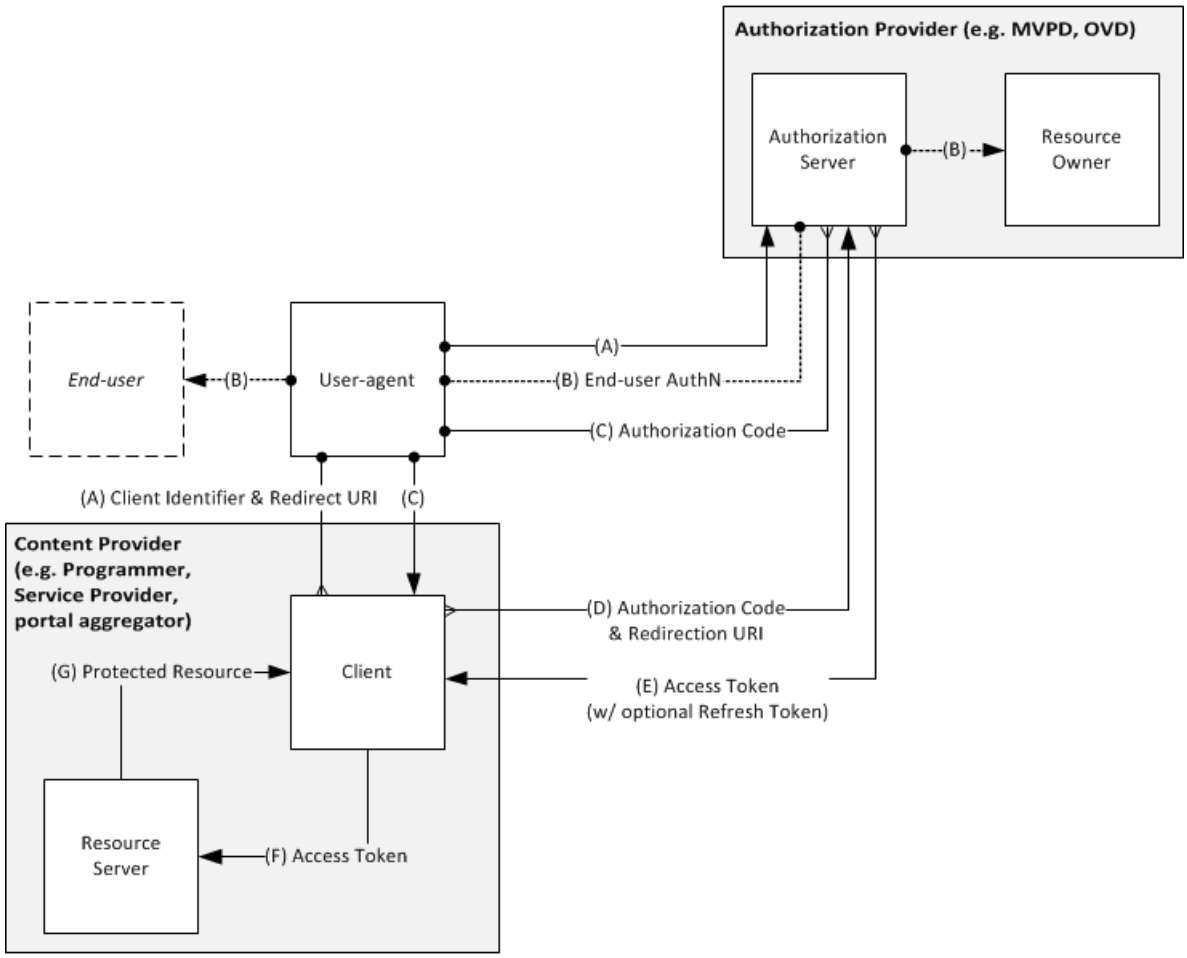

Figure 3 – Web Application Protocol Flow

(B) **End-user Authentication.** The authorization server authenticates the End-user and establishes whether the resource owner grants or denies the client's access request.

Note that in the OMAP use of OAuth 2.0, the Resource Owner is the Authorization Provider (e.g., an MVPD), and that the Authorization Provider or their proxy operates the Authorization Server.

The act of establishing whether the Resource Owner grants or denies the Client's access request will depend on the status of the End-user with the Resource Owner.

End-user Authentication occurs at any point prior to the issuance of an Authorization Code by the AS. The Authorization Server may use a variety of means to maintain an Authentication Session for use in subsequent Authorization Requests, so that repeated End-user Authentications are unnecessary.

The method used for End-user Authentication is out of the scope of this specification. See section 5.3 for additional information.

(C) **Authorization Response**. Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier.

(D) **Access Token Request**. The client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step. When making the request, a Web Application client presents credentials (e.g., a password or some other mechanism) to the authorization server. The client also includes the redirection URI used to obtain the authorization code for verification.

(E) **Access Token Response**. The authorization server will authenticate the client, validate the authorization code, and ensure that the redirection URI received matches the URI used to redirect the client in step (C). If valid, the authorization server responds back to the client with an access token and optional refresh token.

(F) **Protected Resource Request**. The client makes a protected resource request to the resource server by presenting the access token.

(G) **Protected Resource Response**. The resource server validates the access token, and if valid, serves the request.

2.3 NATIVE APPLICATION PROTOCOL FLOW

Like the Web Application protocol flow, the high level architecture consists of interactions between Content Providers, Authorization Providers, Clients and End-users. In the case of a Native Application, the Client is an application running on the End-user device.

Prior to commencing this flow, the Client MAY need to register with the Authorization Server in order to obtain a Client Identifier and optionally a client_secret. This Client Registration process is out of scope for this version of OMAP. See also section 3.1.

Figure 4 summarizes the OMAP OAuth 2.0 Protocol flow for Native Applications. Unlike the Web Application protocol flow, in the Native Application protocol flow the Access Tokens are not necessarily stored on the Content Provider site. They can be stored on the End-user device and/or in a secured cloud storage location.

(A) **Authorization Request.** The client connects to the authorization endpoint. It includes the Client Identifier and requested scope.

(B) **End-user Authentication.** The authorization server authenticates the End-user and establishes whether the resource owner grants or denies the client's access request.

> *See notes on End-user Authentication in the Web Application Protocol Flow, in section 2.2 above.*

(C) **Authorization Response**. If the resource owner grants access, the authorization server provides the client with an authorization code and any local state provided earlier by the client.

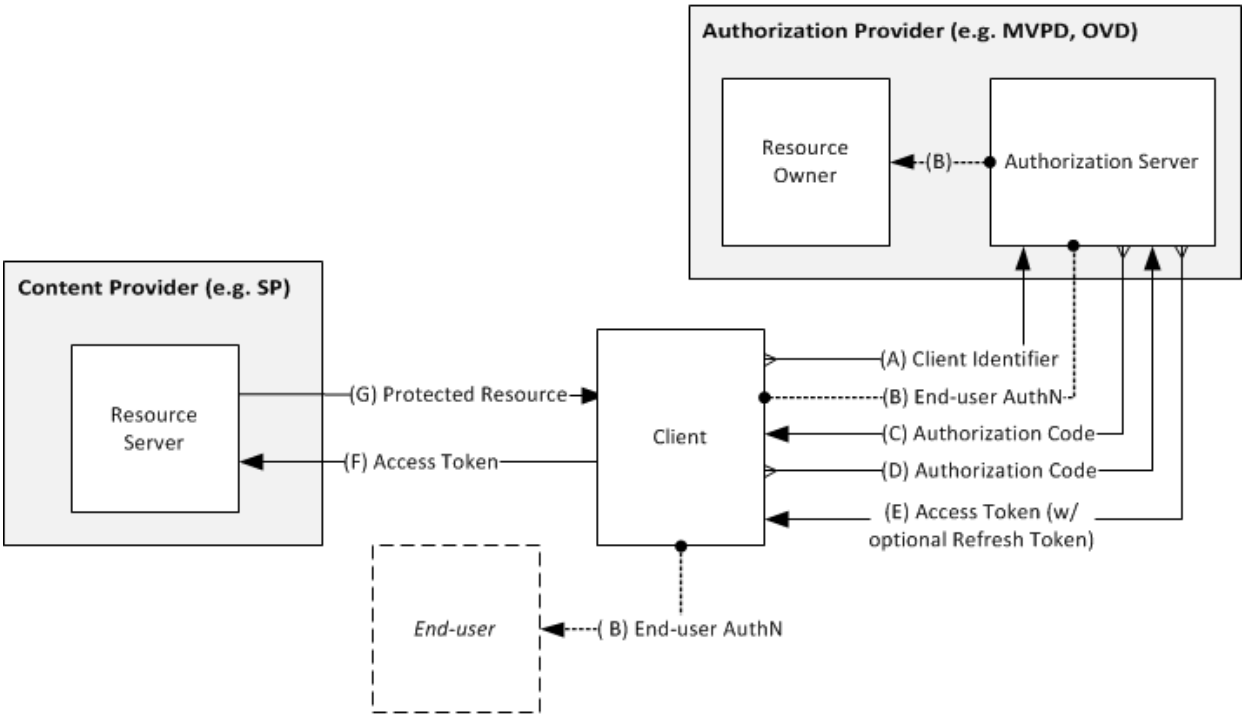

Figure 4 – Native Application Protocol Flow

(D) **Access Token Request**. The client authenticates with the authorization server (see section 3.1) and requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step.

(E) **Access Token Response**. The authorization server authenticates the client and validates the authorization code. If valid, the authorization server responds back with an access token and optional refresh token.

(F) **Protected Resource Request**. The client makes a protected resource request to the resource server by presenting the access token.

(G) **Protected Resource Response**. The resource server validates the access token, and if valid, serves the request.

2.4 ACCESS CONDITIONS AND ACCESS TOKENS

The Authorization Server authorizes the Resource Server to deliver a Protected Resource to the End-user by issuing an Access Token to the Client. Because there are many Protected Resources an End-user may want to access, even within the course of a single End-user session, the Client may request an Access Token that authorizes access either to a particular single Resource, or to a *set* of multiple Resources that share certain characteristics or attributes – for example, all the videos associated with a particular TV channel.

OMAP defines Access Conditions as a character string consisting of name-value pairs that describe a set of Protected Resources along with the syntax of these strings and the circumstances or context in which access to the Protected Resources might occur. An Access

Token is issued for a specific set of Access Conditions. These Access Condition strings are passed via the "scope" parameter in Authorization Requests and Responses, and are also encoded into Access Tokens.

The Authorization Server bases its authorization decision on the Access Conditions specified by the Client in the Authorization Request. The Authorization Server MAY decide, however, to grant access under a different set of Access Conditions than those that were requested. In such cases, the AS specifies the granted Access Conditions in the Authorization Response, and the Client MUST verify that those conditions actually apply to any Protected Resources it may subsequently attempt to access.

It is expected that Clients will cache Access Tokens, along with their associated Access Conditions, so that they may be re-used in multiple Resource requests on behalf of a given End-user. Such re-use of Access Tokens enhances system efficiency, reducing the request volume borne by the Authorization Provider's infrastructure, and improves the End-user experience.

The format of Access Condition strings, and the algorithm by which Access Tokens are selected for re-use, are specified in section 4, below.

3 THE PROTOCOL

In OMAP the OAuth 2.0 Authorization Code grant type is used to obtain both Access and Refresh Tokens (see “4.1 Authorization Code” in [OAUTH]).

3.1 CLIENT AUTHENTICATION

Web Application Clients MUST, and Native Application Clients SHOULD, authenticate with the Authorization Server Token Endpoint as described in 3.2.1 of [OAUTH].

The Client MAY authenticate to the Authorization Server using the HTTP Basic Authentication Scheme [RFC2617], by inserting its password into the request body using the “client_secret” parameter, or by another authentication mechanism supported by the Authorization Server.

Currently Native Applications distributed through a public market may not be capable of maintaining secrets (e.g., a password used to authenticate to the AS). In such a scenario, Clients MAY obtain a password after download and install via a client registration protocol. One option for this is defined by the OpenID Connect Dynamic Client Registration protocol [OICREG].

3.2 AUTHORIZATION REQUEST

The Client constructs the Authorization Request using the Authorization Code grant type as defined in [OAUTH], section 4.1.1.

The “scope” parameter of this request MUST be constructed in accordance with section 3.3 of [OAUTH] and section 4.2 of this specification.

3.3 AUTHORIZATION RESPONSE

If the Authorization Provider acting as the Resource Owner grants the access request, the Authorization Server issues an Authorization Code and delivers it to the Client as defined in [OAUTH], section 4.1.2.

If the authorization is not granted, the Authorization Response SHOULD include Remediation Data. See section 3.10 of this specification.

Note that if authorization is partially granted any Remediation Data provided to the Client MUST be provided in the Access Token Response (see section 3.5).

3.4 ACCESS TOKEN REQUEST

The Client constructs the Access Token Request as defined in [OAUTH], section 4.1.3.

3.5 ACCESS TOKEN RESPONSE

If the Access Token Request is valid and authorized, the Authorization Server issues an Access Token and a Refresh Token as described in [OAUTH], section 4.1.4

If the authorization is only partially granted the Access Token Response SHOULD include Remediation Data. See section 3.10 of this specification.

There MAY be Obligation Data associated with the Access Token Response. See section 3.11.

If the "scope" parameter provided to the AS in the Authorization Request is NULL, the Access Token Response MAY or MAY NOT include an Access Token, but if the End-user has been successfully Authenticated, the response MUST include a Refresh Token.

3.6 REFRESH REQUEST

The Client constructs the Refresh Request as defined in [OAUTH], section 6.

3.7 REFRESH RESPONSE

If valid and authorized, upon receiving a Refresh Request the Authorization Server issues a new Access Token as described in [OAUTH], section 6.

If the refresh is not granted or only partially granted the Refresh Response SHOULD include Remediation Data. See section 3.10 of this specification.

There MAY be Obligation Data associated with the Refresh Response. See section 3.11.

3.8 AUTHORIZATION STATUS REQUEST

There are scenarios in which the Client will want to know whether an Authenticated End-user has permission to access Protected Resources under specific Access Conditions without actually requesting an Access Token.

For example, a Web Application might want to build a customized web page highlighting content the End-user has permission to watch, indicating "locked" content or excluding content based on parental control parameters. To accommodate such scenarios, OMAP provides the Authorization Status Request and Response.

The Authorization Status Request is handled by a Status Endpoint, not the Authorization Endpoint. The Authorization Status Request improves the responsiveness and efficiency of the Client by reducing the number of round trips required as compared to the Authorization Request. The Authorization Status Response does not return an Access Token; it simply indicates the End-user's permissions by returning them in the "scope" parameter of the Authorization Status Response.

Either the Client or the User-agent may issue an Authorization Status Request. The Authorization Status Request URI is constructed by adding the following parameters to the query component of the Status Endpoint URI using the "application/x-www-form-urlencoded" format as defined by [HTML]:

`client_id`	REQUIRED	The Client_id in the Authorization Status Request URI MUST be constructed as described in section 2.3 of [OAUTH].
`scope`	OPTIONAL	The scope in the Authorization Status Request URI MUST be constructed in keeping with section 3.3 of [OAUTH]. Scope MUST contain the queried permissions encoded as described in section 4 of this specification.
`refresh token`	OPTIONAL	A Refresh Token provided in an Access Token Response MUST be included if the Authorization Status Request is being issued by the Client. If there is no Refresh Token available, the Client can direct the User-agent to issue an Authorization Request to the Authorization Server with a NULL “scope” parameter. If the End-user is Authenticated, this MUST result in the Client acquiring a Refresh Token (see sections 3.5 and 4.3.1). The Refresh Token is not required if the request is coming from the User-agent. However, the successful use of the Authorization Status Request by a User-agent depends on the Authorization Server maintaining an End-user Authentication Session.

3.9 AUTHORIZATION STATUS RESPONSE

The Authorization Server’s response to the Authorization Status Request is identical to the Access Token Response, except that the tokens are absent. In particular, the Authorization Status Response MUST include a “scope" field if the End-user permissions being indicated in the response are different from the queried permissions.

The Authorization Status Response MAY include Remediation Data. See section 3.10 of this specification.

The Client MUST NOT interpret an Authorization Status Response as an actual Authorization to access Protected Resources, but rather as an indication of what an Access Token Response would likely be if an Access Token Request were to be made.

3.10 REMEDIATION DATA

When the request to access Protected Resources is not granted or only partially granted, the Authorization Response, the Access Token Response or Refresh Response SHOULD include context and remediation information. The Client MAY use this information to explain to the End-user why full access was not granted, and potentially enable the End-user to remedy the problem.

This information is similar to the OAuth 2.0 error response (see [OAUTH], section 4.1.2.1). OMAP uses a separate error mechanism because in OMAP Access Tokens may be returned with a different scope than requested by the Client, allowing remediation even though there has not been an OAuth 2.0 protocol error.

The JSON response MAY contain Remediation Data consisting of an array of remediation objects ("rem"). Each "rem" contains the following fields:

`type` A string identifying the type of remediation. OMAP provides a pre-defined set of remediation types (see section 5.1). It is RECOMMENDED that any added remediation types be preceded by a URN identifying a namespace.

`msg` (OPTIONAL) A human-readable string that can be displayed to the End-user.

`url` (OPTIONAL) A link to a web page where further action may be taken.

Note that a Native Application without a User-agent MAY implement remediation without benefit of the 'url' field. This requires that the action to be taken for each remediation "type" be communicated by the Authorization Provider to the Native Application developer. For example, if the Native Application is specific to the Content Provider or the Authorization Provider, these remediation 'types' and their corresponding rules might be subject to a bilateral agreement between the Authorization Provider and the Content Provider.

The following informational examples illustrate the use of the OMAP Remediation Data feature. The examples include:

- An Authorization Response that denies access because the End-user device is not registered.
- An Access Token Response that authorizes a subset of the requested Access Conditions due to the parental control setting for the End-user account.
- A Refresh Response that authorizes a subset of the original Access Conditions because the End-user dropped a premium channel from their Authorization Provider subscription agreement.
- A Refresh Response that returns no Access Token because the Authorization Server requires the End-user to re-authenticate.

Note that in all these examples a Native Application without a User-agent could implement the remediations based solely on the "type" field.

3.10.1 REMEDIATION – DEVICE NOT REGISTERED (INFORMATIONAL)

In this example the Authorization Response returns an error (see section 4.1.2.1, [OAUTH]). There is no Authorization Code because the End-user device is not registered with the Authorization Provider.

The Remediation Data provides a means for the End-user to register their device so that the Authorization Request can proceed successfully.

The Remediation Data is returned with the OAuth 2.0 error code (line-breaks added for readability):

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=access_denied
&state=xyz&rem_type=urn:example:register_device&rem_msg=Device%20
not%20registered&rem_url=http://mvpd-example.com/manage/devices
```

3.10.2 REMEDIATION – PARENTAL CONTROL SETTING (INFORMATIONAL)

In this example the Authorization Response was successful, providing an Authorization Code, but the Access Conditions granted are a subset of those requested by the Client, because some of the Protected Resources requested are not available due to the End-user's parental control settings.

The Client sent the following "scope" string in the Authorization Request:

```
"scope":"urn:example:channel=HBO"
```

In the Access Token Response, the "scope" field and the Access Conditions in the Access Token restrict access to content based upon the End-user's maximum parental control setting.

```
"scope":"urn:example:channel=HBO&urn:example:rating=G,PG-13"
```

The Remediation Data provided in the Access Token Response enables the Client to notify the End-user of this restriction and provides the means for the End-user to manage the parental control settings for their account. It consists of a single "rem" object, with a "type" of "urn:oatc:omap:rem:pco" (see section 5.1.3).

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
   "scope":"urn:example:channel=HBO&urn:example:rating=G,PG-13"
   "access_token":" 2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":360000,
   "refresh_token":" tGzv3JOkF0XG5Qx2TlKWIA"
   "rem":
   [
      {
        "type": "urn:oatc:omap:rem:pco",
        "msg" : "Max parental control rating exceeded",
        "url" : "http://mvpd-example.com/manage/parental_controls"
      }
    ]
 }
```

3.10.3 REMEDIATION – CHANNEL NO LONGER AVAILABLE (INFORMATIONAL)

In this Remediation Data example the Refresh Response contains an Access Token providing a subset of the Access Conditions requested because the End-user has dropped a channel from their subscription lineup.

In the original Authorization Request the Client sent the following “scope” string:

```
"scope":"urn:example:channel=CH1,CH2,CH3"
```

The Authorization Response received included this scope, but because of the End-user’s changed channel lineup, the “scope” parameter and the “ac” field of the Access Token returned in the Refresh Response removes access to one of the channels.

```
"scope":"urn:example:channel=CH1,CH2"
```

The Remediation Data provides multiple remediations for this situation. The End-user can acquire rights to the content by either updating their subscription with the Authorization Provider or access the content on a “pay per view” basis.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
   "scope":"urn:example:channel=CH1,CH2"
   "access_token":" 2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":360000,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
```

```
   "rem":
   [
      {
        "type": "urn:oatc:omap:rem:upgrade",
        "msg" : "Would you like to upgrade your account?",
        "url" : "http://mvpd-example.com/manage/upgrade-acct"
      },
      {
        "type": "urn:oatc:omap:rem:ppv",
        "msg" : "Would you like to pay to view?",
        "url" : "http://mvpd-example.com/purchase/ppv"
      }
    ]
}
```

3.10.4 REMEDIATION – RE-AUTHENTICATION REQUIRED (INFORMATIONAL)

In this example the Refresh Response provides no Access Token because the Authorization Server requires the End-user to re-authenticate with the Identity Provider.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
   "error" : "invalid_scope",
   "rem":
   [
      {
        "type": "urn:oatc:omap:rem:reauthn",
        "msg" : "You must sign in to continue to receive content",
        "url" : "http://mvpd-example.com/manage/credentials"
      }
    ]
}
```

3.11 OBLIGATION DATA

When the request to access Protected Resources is granted, either the Access Token Response or the Refresh Response MAY include Client obligation information indicating that the grant was provided conditionally. This information is referred to as the Obligation Data.

The JSON response MAY contain Obligation Data consisting of an array of obligation objects (“obl”). Each “obl” contains the following fields:

type — A string identifying the type of obligation. OMAP provides a pre-defined set of obligation types (see section 5.1). It is RECOMMENDED that any added obligation types be preceded by a URN identifying a namespace.

<other> (optional) Additional fields as required by the obligation type.

Note the conditional obligations associated with the content are subject to a bilateral agreement between the Authorization Provider and the Content Provider, so the types of obligations and additional fields required to support them are not defined in the OMAP specification.

The following is an example of Obligation Data instructing the Client to log the authorization.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token":"<access token>",
  "token_type":"example",
  "expires_in":360000,
  "refresh_token":"<refresh token>"
  "obl":
  [
     {
        "type" : "urn:oatc:omap:obl:log",
     }
  ]
```

4 ACCESS CONDITIONS, SCOPE, AND ACCESS TOKENS

In its request to the Authorization Server for an Access Token, the Client includes the Access Conditions, passing them in the "scope" parameter of the Authorization Request. Access Conditions describe the Protected Resources that the Client would like to access and the circumstances or context in which the access will occur.

The Authorization Server MAY issue an Access Token granting access under exactly the Access Conditions the Client requests, or under different conditions. If the granted conditions are different from the requested conditions, the Authorization Server MUST include a "scope" field in the Response, indicating the Access Conditions actually granted.

Whether the granted conditions are the same as the requested conditions or different, they are included in the JWT Claims Set contained within the returned Access Token.

4.1 ACCESS TOKEN FORMAT

OMAP Access Tokens are JSON Web Tokens (see [JWT]). A JWT is a string encoding a JSON data structure - the JWT Claims Set. An OMAP JWT MAY be encrypted and SHOULD be digitally signed. If used, encryption MUST be done using JSON Web Encryption [JWE,JWA] and digital signing MUST be done using JSON Web Signature [JWS,JWA].

Digitally signing the Access Token provides assurance that attempts to modify the contents of the Access Token can be detected by the Resource Server.

The Authorization Server MAY also encrypt the Access Token. In this case, the Authorization Server MAY use the SPID name-value pair provided in the “scope” parameter of the Authorization Request to determine what key to use when producing the JSON Encrypted Key from a Content Encryption Key [JWE].

An encrypted Access Token can only be used by the Service Provider with access to the key needed to decrypt the JSON Encrypted Key, so encryption is one means of restricting the use of the Access Token to a single Service Provider.

Restricting the use of the Access Token to a single Server Provider may also be accomplished by placing the Service Provider Identifier (SPID) value provided in the Authorization Request in the “aud” field of the Access Token JWT Claims Set (see Table 2), and digitally signing the Access Token [JWS,JWA].

Specifying an SPID in the “aud” field restricts the use of the Access Token to a single SP because a Resource Server MUST NOT use an Access Token to provide access to Protected Resources if the “aud” field is present and the Service Provider hosting the Resource Server is not the one indicated in the “aud” field of the OMAP JWT Claims Set.

The following is an example of an OMAP JWT Claims Set (see Table 2):

```
{
 "iss" : "mvpd1",
 "exp" : 1340236800,
 "user":
   {
    "id": "3f7b3dcf-1674-4ecd-92c8-1544f346baf8",
    "persistent": true,
    "breadth": "sp1 sp2",
     "account":
       {
        "id": "3f7b3dcf-1674-4ecd-92c8-1544f346baf8",
        "persistent": true,
        "breadth": "sp1"
        },
   },
 "ac":"urn:example:channel=HBO&spid=sp1"
}
```

The contents of the OMAP JWT Claims Set are defined in Table 2.

Table 2 – OMAP JWT Claims Set Fields

Field	Type	Definition
"iss" :	*string*	(REQUIRED) Issuer. Reference for the Authorization Provider. Syntax: StringOrURI (see section 4.1 of [JWT]).
"aud" :	*string*	(OPTIONAL) Audience. Reference for the Service Provider. Syntax: StringOrURI (see section 4.1 of [JWT]). If absent, the Resource Server MUST treat the audience for the JWT Claims Set as unrestricted, allowing it to be used by any Service Provider. The Resource Server MUST reject an Access Token if the "aud" field is present and does not correspond to a Service Provider using that Resource Server. If the Authorization Server wants to restrict use of the Access Token to the Service Provider hosting the requested Protected Resources, it SHOULD insert the SPID from the "scope" parameter of the Authorization Request into the "aud" field. This enables the Resource Server to determine whether the Access Token

Field	Type	Definition
		applies to the current Service Provider without parsing the "ac" field.
"exp" :	*number*	(REQUIRED) Expiration date/time of Authorization. Syntax: IntDate (see section 4.1 of [JWT]).
"user" : {	*object*	(REQUIRED) The individual making the request.
"id" :	*string*	A string that contains an End-user identifier. Syntax: String (see section 4.1 of [JWT]).
"persistent" :	*true or false*	(OPTIONAL) A Boolean indicating whether the user ID is permanent or transient. Default value is TRUE. Note that this field is similar in function to the SAML 2.0 nameid-format:persistent value.
"breadth" :	*string*	(OPTIONAL) A space-delimited string or URI reference designating one or more entities where the user ID is relevant. If absent, the breadth is unrestricted. It is RECOMMENDED that "breadth" use the same syntax as "aud". Syntax: String (see section 4.1 of [JWT]).
"account" : {	*object*	(OPTIONAL) A master account holder. This MAY be the same as the "user" and/or subsume another user (sub-account). The Default is the user field.
"id" :	*string*	A string that contains the Account identifier. Syntax: String (see section 4.1 of [JWT]).
"persistent" :	*true or false*	(OPTIONAL) A Boolean indicating whether the user ID is permanent or transient. Default value is TRUE. Note that this field is similar in function to the SAML nameIDformat field.
"breadth" :	*string*	(OPTIONAL) A space-delimited string or a URI reference designating one or more entities where the user ID is relevant. If absent, the breadth is unrestricted.

Field	Type	Definition
		Syntax: String (see section 4.1 of [JWT]). It is RECOMMENDED that "breadth" use the same syntax as "aud". Note that this field is similar in function to the SAML nameQualifier field.
}, **},**		
"ac" :	*string*	(REQUIRED) A string containing the Access Conditions. See section 4.2 for additional information. Syntax: String (see section 4.1 of [JWT]).

4.2 ACCESS CONDITIONS SYNTAX

Access Conditions are expressed using name-value pairs. Each pair is a statement of fact about an access to a Protected Resource. For example, the following condition applies to any access to content from the TV provider HBO:

```
urn:example:channel=HBO
```

Multiple name-value pairs may be compounded using "&", as follows:

```
urn:example:channel=HBO&urn:example:rating=TV-Y
```

These Access Conditions represent the logical intersection of Resources for which the provider is HBO *and* the rating is TV-Y.

The logical union of multiple conditions is expressed using the space character, as follows:

```
urn:example:channel=HBO urn:example:channel=MTV
```

These Access Conditions Subsets together describe the set of Resources for which the channel is either HBO or MTV.

When the Access Condition consists of the logical union of multiple conditions, each component condition is called an Access Condition Subset, or simply a Subset.

A more concise way of expressing logical unions is to specify multiple values separated by commas:

```
urn:example:channel=HBO,MTV
```

Instead of describing a set of multiple Resources, an Access Condition may describe an individual Resource, resulting in an Access Token that only enables access to that particular Resource. For example:

```
urn:example:movie=10.5240/C840-E543-A58F-5C59-1B1C-T
```

4.2.1 CONTEXT IN ACCESS CONDITIONS

Access Conditions may describe aspects of the environment or context in which access to a Protected Resource might occur, such as the geographic location of the End-user, or the type of device or Internet connection being used. For example, the following condition describes access to HBO content, but only from the New York DMA:

```
urn:example:channel=HBO&urn:example:dma=501
```

4.2.2 DEVICE BINDING IN ACCESS CONDITIONS

When sending an Authorization Request, the Client SHOULD include in the ”scope” parameter of the Authorization Request the Device Binding ID (DBID) and the Device Binding Type (DBT).

The Device Binding ID is a string designating the device or domain of devices associated with the Client. The Device Binding Type is a string designating the method used in creating the Device Binding ID. The format is:

```
urn:oatc:omap:dev:dbid=device_id&urn:oatc:omap:dev:dbt=method
```

For the privacy of the End-user, it is RECOMMENDED that a hash of the DBID be used for Authorization Requests. Note that this implies that the DBID SHOULD NOT be used as a Device Management ID (see Appendix 5.2).

The Authorization Server MAY set the TTL on the Access Token based upon the Device Binding Type employed by the Client.

The Authorization Server MAY use the DBID to bind an Access Token to a device or a domain of devices. The Authorization Server MAY need to encrypt the Access Token to make this binding secure, depending upon the security of the Device Binding Type employed by the Client.

See sections 5.2.2 and 4.3.1 for additional information.

4.2.3 SERVICE PROVIDER IN ACCESS CONDITIONS

When sending an Authorization Request the Client MUST include a Service Provider Identifier (SPID) in the ”scope” parameter of the Authorization Request.

The SPID is either a string or a URI representing the Service Provider for the Protected Resources requested. The format for expressing the SPID value in the Access Conditions is:

```
urn:oatc:omap:aud:spid=sp1
```

See section 4.3.1 for additional information.

4.2.4 NAMESPACES IN NAME-VALUE PAIRS

To avoid ambiguity and promote interoperability, the names used in Access Condition name-value pairs SHOULD be URNs that identify the namespaces from which their corresponding values are taken.

There are numerous database services in use that provide standardized, structured metadata for multimedia content. Similarly, there are multiple content rating systems available for the purpose of providing parental guidelines and control. OMAP provides a mechanism for communicating these types of metadata and their associated namespaces, but is neutral regarding the use of specific database service providers or ratings systems; the Content Provider and Authorization Provider will need to mutually agree which of these systems to use, or implement mappings between different systems and/or providers.

Notwithstanding the foregoing, OMAP necessarily defines several protocol-specific namespaces. These are distinct from and non-overlapping with the aforementioned multimedia metadata namespaces. These OMAP-defined namespaces are detailed in section 5.1.

4.3 PROCESSING MODEL FOR ACCESS CONDITIONS

Access Conditions may be narrow, referring to as few as one individual Protected Resource, or broad, referring to a large set of Resources. Access Tokens that are issued with broad Access Conditions have the potential to be re-used to access multiple Resources over a period of time, resulting in better system performance than if every Resource request required a new Access Token. To achieve such re-use, Clients and Authorization Servers MUST construct Access Conditions according to the normative rules detailed below.

4.3.1 NORMATIVE RULES

1. The Client MUST specify Access Conditions in sufficient detail, using enough name-value pairs that the Authorization Server recognizes, to allow the Authorization Server to apply relevant business rules and issue its authorization decision. Agreement concerning recognized name-value pairs and the degree of detail that will be required in Access Conditions is outside the scope of this specification.
2. The Client MAY include name-value pairs that are not understood by the Authorization Server. The Authorization Server MUST ignore any such pairs, rather than treat them as an error.
3. The Client MAY specify Access Conditions that describe access to a single Protected Resource, but the Client SHOULD specify Access Conditions that are broader (describing a larger set of Protected Resources) if it is likely that the resulting Access Token would be able to be re-used in the future.
4. The Client MAY provide a NULL “scope” parameter. The Authorization Server MUST NOT treat this as an error. In this case the Authorization Server MAY or MAY NOT return an Access Token, but the Authorization Server MUST return a Refresh Token.

5. The Authorization Server MUST grant access to a narrower set of Resources than requested by the Client, if required by business rules, rather than treat the request as an error. The Authorization Server MUST eliminate from the requested Access Conditions any Access Condition Subsets for which it refuses to grant access, and return the narrower Access Conditions in its response. Any Subsets not eliminated MUST appear exactly as they were requested by the Client, and in the same order, at the beginning of the granted Access Conditions.
6. The Authorization Server SHOULD grant Access Conditions that are as broad as business rules allow, by adding Subsets to the requested Access Conditions. Any added Subsets MUST appear after all requested Subsets in the granted Access Conditions.
7. When the Client presents the Access Token to the Resource Server, if the DBID and DBT were submitted in the Authorization Request, the Client MUST re-acquire the DBID and DBT from the device and present them with the Access Token to the Resource Server.
8. The Resource Server MUST reject an Access Token if the "dbid" and "dbt" fields of the JWT Claims Set are non-NULL and they do not match the DBID and DBT values presented by the Client to the Resource Server (see sections 5.2.2 and 5.5).

4.3.2 EXTENDED EXAMPLE (INFORMATIONAL)

The following extended example demonstrates the application of these normative rules by the Client and Authorization Server to negotiate Access Conditions and achieve the re-use of Access Tokens. For brevity, this example makes use of abstract name-value pairs.

1. The Client wishes to access a Protected Resource, R_1. It forms an Access Conditions string, AC_1, describing that Resource and the conditions under which the access will occur:

   ```
   n1=v1&n2=v2&n3=v3
   ```

2. In order to increase the likelihood that the Authorization Server will grant access to a broader set of Access Conditions than AC_1, thus decreasing the likelihood that new Access Tokens will need to be requested in the future, the Client MAY form a broader Access Conditions string, $AC_1^{request}$. The Client then sends $AC_1^{request}$ via the "scope" parameter of the Authorization Request:

   ```
   n1=v1&n2=v2&n3=v3 n1=v1 n2=v2 n3=v3
   ```

3. The Authorization Server applies business rules and determines that the requested Access Conditions $AC_1^{request}$ cannot be granted in full, but that a subset of them can be granted. It forms a new Access Conditions string, $AC_1^{response}$, encodes it into an Access Token AT_1, and returns it via the "scope" field along with the Access Token in the Access Token Response:

   ```
   n1=v1&n2=v2&n3=v3 n1=v1 n2=v2
   ```

4. The Client verifies that the granted Access Conditions, $AC_1^{response}$, are a superset of the original conditions, AC_1, required to access Resource R_1. This verification can be performed simply and quickly, because $AC_1^{response}$ contains AC_1 as a prefix.

5. The Client requests Resource R_1 from the Resource Server, passing both the Access Token AT_1 and the Access Conditions $AC_1^{request}$ in addition to any other arguments the Resource Server may require. The Resource Server MAY consider some, all, or none of the Access Conditions $AC_1^{request}$ reported by the Client to validate the Access Token.

6. The Client stores Access Token AT_1 in its token cache, keyed by its associated Access Conditions $AC_1^{response}$.

7. At a later time, the Client wishes to access another Resource, R_2, and forms a new Access Conditions string, AC_2, to describe it:

```
n1=v1&n2=v4&n3=v5
```

8. The Client locates Access Token AT_1 in its token cache, and detects that its associated Access Conditions, $AC_1^{response}$, are a superset of AC_2, because of the portion shown in bold below:

```
n1=v1&n2=v2&n3=v3 n1=v1 n2=v2
```

9. Provided that the TTL associated with the cached token has not expired, the Client does not request a new Access Token from the Authorization Server. Instead, the Client requests Resource R_2 from the Resource Server, passing the Access Token AT_1 and the Access Conditions AC_2. The Resource Server verifies that the Access Conditions $AC_1^{response}$ embedded in AT_1 are a superset of AC_2, and thus grants the request.

10. At a later time, the Client wishes to access another Resource, R_3, and forms a new Access Conditions string, AC_3, to describe it:

```
n1=v1&n2=v6&n3=v7
```

11. The Client determines that AC_3 is not a subset of $AC_1^{response}$, and therefore that AT_1 does not authorize access to R_3. The Client thus issues a Refresh Request, passing AC_3 via the “scope” parameter.

12. The Authorization Server responds with Access Token AT_2 and Access Conditions $AC_3^{response}$:

```
n1=v1&n2=v6&n3=v7 n1=v1 n2=v6
```

13. The Client uses AT_2 to request R_3 from the Resource Server, and stores AT_2 keyed by $AC_3^{response}$ in its token cache.

4.4 TOKEN VERIFICATION AND CACHING

Before attempting to access a Protected Resource, the Client MUST acquire an Access Token whose Access Conditions encapsulates those required to access that Resource. Similarly, the Resource Server MUST verify that an attempt to access a Protected Resource is authorized by the presented Access Token.

The set logic expressed by the Access Condition syntax implies a simple algorithm for verifying that one set of Access Conditions is a superset of another. This algorithm is documented in Appendix 5.4.

4.4.1 TOKEN VERIFICATION BY CLIENTS

When the Client constructs Access Conditions describing access to a particular Resource, requests them from the Authorization Server via the "scope" parameter, and receives in response either no "scope" field or one whose value contains the requested Access Conditions as a prefix, then the Client SHOULD assume that the corresponding Access Token authorizes access to the desired Resource, without any further processing.

Otherwise, the Client MUST verify that the granted Access Conditions are a superset of those required to access the Protected Resource, using an algorithm like the one in Appendix 5.4.

4.4.2 TOKEN VERIFICATION BY RESOURCE SERVERS

The Resource Server MUST verify that the Access Token presented with a Resource Request contains Access Conditions, in the “ac” field of the embedded OMAP JWT Claims Set, that are a superset of those required to access the Resource. It MAY do so by constructing a new Access Conditions string describing the current Resource Request, and then using an algorithm like the one in Appendix 5.4 to verify that those Access Conditions are a subset of the Access Conditions in the Access Token.

4.4.3 TOKEN CACHING BY CLIENTS

The Client SHOULD maintain a cache of Access Tokens, and use it to avoid requesting Access Tokens more frequently than necessary. Before requesting an Access Token for a given set of Access Conditions, the Client SHOULD check its cache to see if a usable, unexpired Access Token has already been obtained.

Each entry in an Access Token cache SHOULD contain, at minimum, an Access Token (or a “negative” response as discussed in Section 4.4.7 below) and the Access Conditions, TTL and any Remediation or Obligation Data that correspond to that Access Token.

4.4.4 UNINDEXED SET (INFORMATIONAL)

Cache entries may be stored in a **set**, with duplicates eliminated by comparing their Access Condition strings. The set may be searched linearly using an algorithm like that found in Appendix 5.4.

4.4.5 MAP INDEX (INFORMATIONAL)

Cache entries may be stored in a **map**, keyed by the space-separated Subsets of their Access Conditions. For every Subset in the Access Conditions a separate entry in the map is created with the Subset as key to that entry.

This method can be effective for Clients that "broaden" their desired Access Conditions by adding Subsets to them before making the Authorization Request. However, unless this is done fully and consistently, this indexing method can fail to locate Access Tokens that do, in fact, authorize a given set of Access Conditions.

4.4.6 TRIE INDEX (INFORMATIONAL)

The algorithm in Appendix 5.4 uses a data structure called a **trie** to determine whether one set of Access Conditions is a superset of another. This same data structure can be used to index Access Tokens according to their Access Conditions. Because the structure of the index mirrors the structure of Access Conditions syntax itself, looking up a set of Access Conditions in the index is fully equivalent to checking every entry in the cache, but faster. Unlike with the map index, above, failure to locate an entry implies that the cache contains no usable Access Token.

4.4.7 NEGATIVE CACHING

As discussed in Section 4.3.1, the Authorization Server MUST eliminate from the Access Conditions it returns to Clients any Subsets of the requested Access Conditions to which it is unwilling to grant access. The omission of these Subsets SHOULD be used by the Client to maintain "negative" entries in its Access Token cache, keyed by the omitted Subsets.

It is RECOMMENDED that the TTL associated with a negative entry in the Access Token cache be a carefully chosen, small value and not a value was possibly received in an Access Token or Refresh Token Response. Choosing a TTL value for a negative entry SHOULD balance the need to reduce request volume to the Authorization Server with the need to provide the End-user timely access to recently acquired rights.

In addition, when the Authorization Server denies the entire request and provides no Access Token, this denial SHOULD be used by the Client to place a "negative" entry into the Access Token cache, keyed by the denied Access Conditions.

When the Client wants to access a Resource, and can find nothing in the cache whose Access Conditions are a superset of that Resource except those with "negative" responses associated with them, the Client SHOULD conclude that the access will be denied until that cache entry expires, and SHOULD NOT make a request for an Access Token.

5 APPENDICES

5.1 OMAP NAMESPACE

The Namespace ID for OATC specifications is "OATC". URNs assigned by OATC will have the following hierarchical structure.

```
urn:oatc:{standard}:{string}
```

Where:

`standard`	Is a US-ASCII string that conforms to URN Syntax requirements ([RFC2141]) and corresponds to the name of OATC specifications
`string`	Is a US-ASCII string conforming to URN Syntax requirements ([RFC2141])

The container for the OMAP namespace is `urn:oatc:omap`.

5.1.1 ACCESS TOKEN AUDIENCE IDENTIFIERS

`urn:oatc:omap:aud`	Identifiers specific to the audience for the Access Token.
`urn:oatc:omap:aud:spid`	Service Provider identifier. See sections 4.1 and 4.2.3.

5.1.2 DEVICE BINDING IDENTIFIERS

`urn:oatc:omap:dev`	Identifiers for the device or domain of devices associated with the authorization.
`urn:oatc:omap:dev:dbid`	Device Binding ID. See sections 4.2.2, 4.3.1 and 5.2.
`urn:oatc:omap:dev:dbt`	Device Binding Type. See sections 4.2.2, 4.3.1 and 5.2.

5.1.3 REMEDIATION IDENTIFIERS

`urn:oatc:omap:rem`	Identifiers for Remediations. See sections 3.3, 3.5, 3.7, 3.9, 3.10 and 4.4.3.
`urn:oatc:omap:rem:upgrade`	The End-user MAY upgrade a subscription to gain broader access.
`urn:oatc:omap:rem:ppv`	The End-user MAY pay for one-time access.
`urn:oatc:omap:rem:pco`	The End-user MAY override a parental control restriction.
`urn:oatc:omap:rem:reauthn`	The End-user MAY re-authenticate to gain access to the content.

5.1.4 OBLIGATION IDENTIFIERS

`urn:oatc:omap:obl`	Identifiers for Obligations. See sections 3.5, 3.7, 3.11 and 4.4.3.
`urn:oatc:omap:obl:log`	The Client MUST log the AuthZ transaction.
`urn:oatc:omap:obl:reauthn`	The Client must acquire a new Refresh Token.

5.2 DEVICE IDENTIFICATION (INFORMATIONAL)

There are three types of device identification which users of the OMAP specification are likely to encounter. These are:

Device Management ID (DMID)	A device identifier used by an Authorization Provider to securely identify, count, limit and manage devices associated with an End-user account.
Device Binding ID (DBID)	A device identifier which can be included by the Authorization Server in an Access Token, effectively "binding" the Access Token to a device or a domain of devices.
DRM ID	A device identifier used by a digital rights management system.

The OMAP specification defines normative use of a Device Binding ID and provides recommendations on the use of a Device Management ID.

Figure 5 illustrates the relationship of these identifiers and, in the case of the Device Binding ID and the Device Management ID, their intended use.

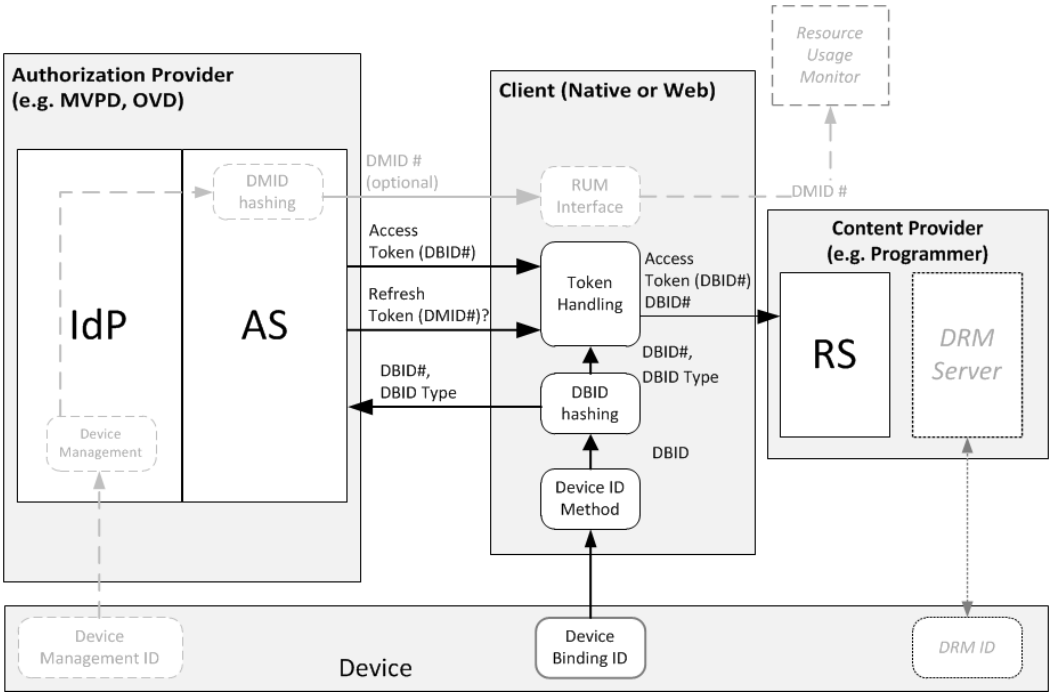

Figure 5 – Device Identification and the OMAP Protocol

5.2.1 THE DEVICE MANAGEMENT ID

If a device registration function is supported by the Authorization Provider's IdP, a Device Management ID can be passed to the Authorization Provider's AS and included in the Refresh Token provided to the Client in the Access Token Response and Refresh Response. In this way the Refresh can be verified by the Authorization Server as being for a device registered to the End-user.

Note that since the DMID is created and used by the IdP-AS, there is no interoperability issue between Clients, AS and RS to compel OMAP to normatively define the DMID.

A likely use for a Device Management ID would be in conjunction with a Resource Usage Monitor in order to monitor concurrent stream usage related to an End-user's account

5.2.2 THE TOKEN BINDING

OMAP defines a Device Binding ID and a Device Binding Type.

When the Client forms an Authorization Request (see section 3.2) or an Authorization Status Request (see section 3.8) it can acquire a DBID from the End-user device. The DBID and string designating the method used to acquire the DBID are provided in the Scope parameter associated with the request sent to the Authorization Server (see section 4.3.1).

5.3 INTEROPERABILITY WITH END-USER AUTHENTICATION (INFORMATIONAL)

This appendix outlines three approaches for making OMAP interoperable with End-user Authentication methods. It includes two approaches for implementing SAML SSO authentication with OMAP Authorization, and an approach for using OMAP on devices without easy data-entry methods (e.g., game consoles or media hubs).

- SAML-OAuth 2.0 Independence
- SAML-OAuth 2.0 Exchange
- Limited device Authentication and Authorization

5.3.1 SAML-OAUTH 2.0 INDEPENDENCE

The Authorization Provider's SAML Identity Provider (IdP) is co-located with the Authorization Server. The Authorization Server acts as a SAML Service Provider endpoint. An Authorization Request results in a redirect to the IdP. The SAML response delivers a SAML assertion to the Authorization Server, which results in OMAP tokens being issued to the Client.

The SAML-OAuth 2.0 independence model is preferable because it places no requirements on the Content Provider to implement a SAML endpoint.

Figure 6 illustrates the protocol flow.

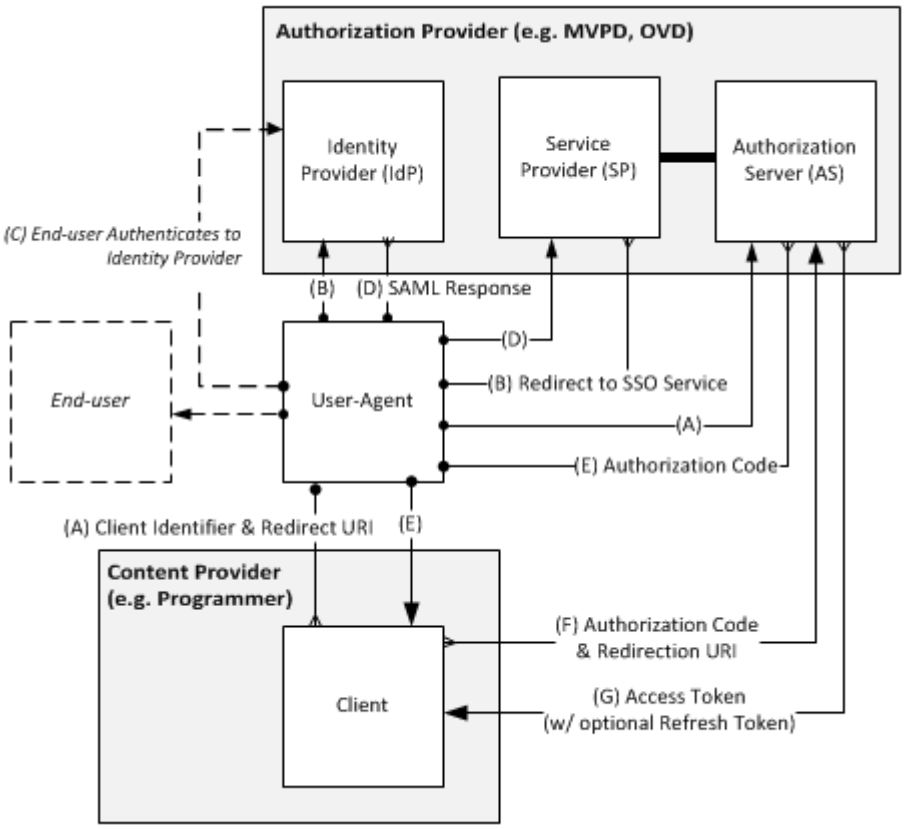

Figure 6 – OMAP SAML SSO Interoperability - Independence

(A) **Authorization Request**. The OMAP authorization request protocol step is independent of the method used for End-user Authentication. In this case, the requester has not yet authenticated, so the AS would redirect to its internally hosted SP in order to generate the AuthN request.

(B) **Redirect to SSO Service**. The User Agent is first redirected to the Authorization Providers' SAML Service Provider (SP) endpoint which makes a SAML AuthN request to the SAML IdP, while also redirecting the User Agent to the IdP's login page.

(C) **End-user Authenticates to Identity Provider.** The End-user enters their credentials and submits the login form.

(D) **SAML Response.** Once the credentials are validated, the IdP issues a SAML AuthN response to the SP and redirects the User Agent to the SP URL.

(E) **Authorization Code Response.** The SP delivers the SAML Assertion to the Authorization Server. Once validated, the Authorization Server issues the Authorization Code in an OMAP Authorization Response. From this point forward the protocol flow is completely independent from how the end user authenticated to the Authorization Provider.

Note - Steps F and G in Figure 6 follow standard OMAP protocol.

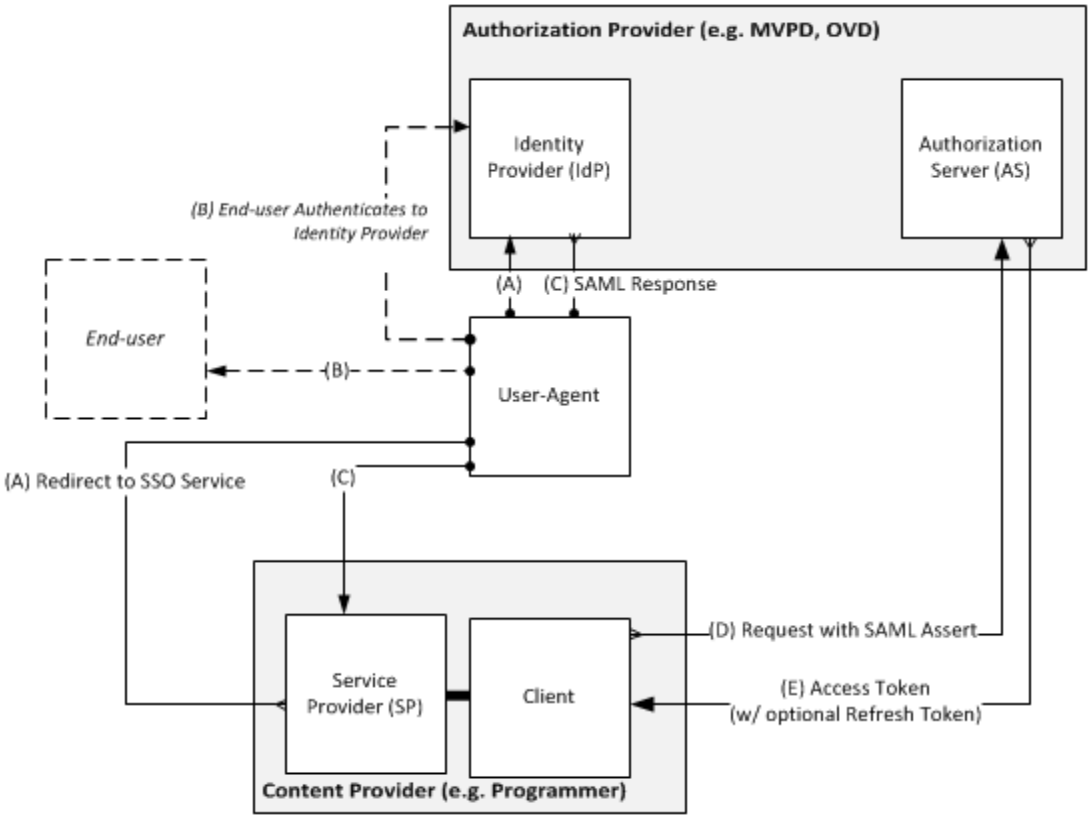

Figure 7 – OMAP SAML SSO Interoperability – Exchange

5.3.2 SAML-OAUTH 2.0 EXCHANGE

In this model the Content Provider implements a SAML Service Provider endpoint. The Content Provider initiates a redirect to the SAML SSO service. The SAML assertion returned to the Content Provider is then exchanged for an Access Token and optionally a Refresh Token.

Figure 7 above illustrates the protocol flow.

(A) **Redirect to SSO Service**. The User Agent loads the Authorization Providers' SAML Service Provider (SP) which makes a SAML AuthN request to the SAML IdP, while also redirecting the User Agent to the IdP's login page.

(B) **End-user Authenticates to Identity Provider.** The End-user enters their credentials and submits the login form.

(C) **SAML Response.** Once the credentials are validated, the IdP issues a SAML AuthN response to the SP and redirects the User Agent to the SP URL.

(D) **OAuth Request with SAML Assertion.** The client includes the SAML Assertion in the Access Token Request.

(E) **Access Token (w/ optional Refresh Token)** The AS validates the SAML Assertion before issuing an Access Token and optional Refresh Token in response (See [OAUTHSML]).

Note that in this case the AS would need to be customized to handle the SAML assertion correctly.

5.3.3 LIMITED DEVICE AUTHENTICATION AND AUTHORIZATION

OAuth 2.0 provides a user delegation flow to deal with devices with limited input capabilities, such as gaming devices and media hubs, when the End-user has access to a User-agent on a more input-capable device, such as a laptop or smart phone. In OAuth 2.0, this type of authorization is called a "Device Flow" (see section 3.7, [OAUTH]).

Figure 8 shows an OMAP limited device flow. Contrast this flow to the Native Application flow given in Figure 4 and the description given in section 3.7 of [OAUTH].

(A) **Authorization Request.** The Client connects to the Token Endpoint. It includes the Client Identifier and requested scope in an HTTP POST constructed as described in 3.7.1 of [OAUTH].

(B) **Authorization Response.** The Authorization Server responses with a "verification code", "verification URI" and "user code", providing these to the Client in the HTTP response body.

(C) **Client prompts the End-user.** The Client presents the "verification URI" and "user code" to the End-user.

(D) **End-user Authentication.** Using the "verification URI", the End-user authenticates with the Authorization Server, providing the AS the "user code". This is a pass-through Authentication to the Resource Owner, who will verify the status of the End-user with the Authorization Provider.

> *See notes on End-user Authentication in the Web Application Protocol Flow, in section 2.2 above.*

(E) **Access Token Request**. While the End-user authenticates with the Authorization Provider and the Resource Owner authorizes (or denies) the Client request, the Client repeatedly polls the Authorization Server to find out when the Authorization step (D) has completed.

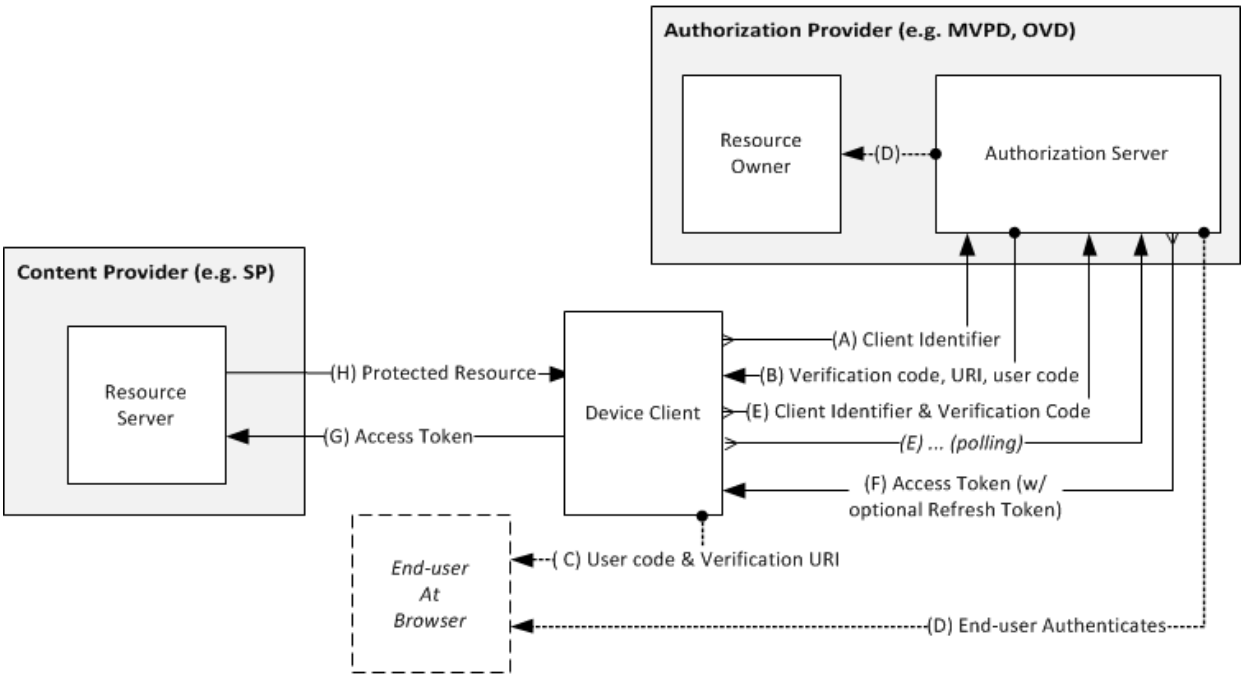

Figure 8 – OMAP Limited Device Flow

(F) **Access Token Response.** If approved, the Authorization Server responds back with an Access Token and the optional Refresh Token (see section 3.7.2 of [OAUTH]).

(G) **Protected Resource Request**. The Client makes a Protected Resource request to the Resource Server by presenting the Access Token.

(H) **Protected Resource Response**. The Resource Server validates the Access Token, and if valid, serves the request.

5.4 ACCESS CONDITION SUBSET/SUPERSET RELATIONSHIP (INFORMATIONAL)

The following Java class demonstrates an algorithm that tests whether a given set of Access Conditions is a subset of another.

```
import java.util.*;

/**
 * AccessConditions encapsulates Access Condition strings, including the
 * algorithm for determining whether one set of Access Conditions is authorized
 * by -- i.e., is a subset of -- another.
 *
 * This algorithm uses a trie ({@link http://en.wikipedia.org/wiki/Trie}) in
 * which the keys are the Access Condition string's space-separated
 * subsets.  All possible permutations of each subset's name-value pairs are
 * generated and indexed separately, so that the result is correct
 * no matter how name-value pairs are ordered.
 *
 * @author boyerb
 *
 */
public class AccessConditions
{
     private String[] mSubsets;

     public AccessConditions(String accessConditions)
     {
          mSubsets = accessConditions.split(" ");
     }

     public String[] getSubsets()
     {
          return mSubsets;
     }

     private class TrieNode
     {
          HashMap<String, TrieNode> mChildren;
          boolean mValue;

          TrieNode() {};

          void put(String accessConditions)
          {
               put(accessConditions.split("&"));
          }
```

```
void put(String[] pairs)
{
     if (mChildren == null)
          mChildren = new HashMap<String, TrieNode>();

     for (int i = 0; i < pairs.length; i++) {
          String[] expandedPairs = expandPairOnCommas(pairs[i]);

          for (int j = 0; j < expandedPairs.length; j++) {
               String pair = expandedPairs[j];
               TrieNode child = mChildren.get(pair);

               if (child == null) {
                    child = new TrieNode();
                    mChildren.put(pair, child);
               }

               if (pairs.length == 1)
                    child.setValue(true);
               else {
                    String[] childPairs = new String[pairs.length - 1];
                    for (int k = 0, l = 0; k < pairs.length; k++)
                         if (k != i)
                              childPairs[l++] = pairs[k];
                    child.put(childPairs);
               }
          }
     }
}

boolean get(String accessConditions)
{
     String[] pairs = accessConditions.split("&");
     return get(pairs);
}
```

```
          boolean get(String[] pairs)
          {
               if (mValue == true)
                    return true;

               for (int i = 0; i < pairs.length; i++) {
                    boolean pairResult = true;

                    String[] expandedPairs = expandPairOnCommas(pairs[i]);

                    int j;
                    for (j = 0; j < expandedPairs.length; j++) {
                         String pair = expandedPairs[j];
                         TrieNode child = mChildren.get(pair);

                         if (child == null)
                              break;
                         else
                              if (pairs.length == 1) {
                                   if (!child.getValue())
                                        break;
                              }
                              else {
                                   String[] childPairs = new String[pairs.length -
1];

                                   for (int k = 0, l = 0; k < pairs.length; k++)
                                        if (k != i)
                                             childPairs[l++] = pairs[k];
                                   if (!child.get(childPairs))
                                        break;
                              }
                    }

                    if (j < expandedPairs.length)
                         pairResult = false;

                    if (pairResult)
                         return true;
               }

               return false;
          }

          public void setValue(boolean value) { mValue = value; }
          public boolean getValue() { return mValue; }
     }
```

```
    public boolean isSubsetOf(AccessConditions superset)
    {
        TrieNode trie = new TrieNode();
        String supersetConditions[] = superset.getSubsets();

        for (int i = 0; i < supersetConditions.length; i++)
            trie.put(supersetConditions[i]);

        for (int i = 0; i < mSubsets.length; i++)
            if (!trie.get(mSubsets[i]))
                return false;

        return true;
    }

    public static String[] expandPairOnCommas(String pairString)
    {
        String[] pairs;

        if (pairString.indexOf(",") == -1) {
            pairs = new String[1];
            pairs[0] = pairString;
        }
        else {
            String[] pair = pairString.split("=");
            String name = pair[0];

            String value = "true";
            if (pair.length > 1)
                value = pair[1];

            String[] values = value.split(",");
            pairs = new String[values.length];

            for (int i = 0; i < values.length; i++) {
                StringBuffer newPairString = new StringBuffer();
                newPairString.append(name).append("=").append(values[i]);
                pairs[i] = newPairString.toString();
            }
        }

        return pairs;
    }
}
```

5.5 CLIENT-RESOURCE SERVER INTERACTION (INFORMATIONAL)

The OMAP Specification does not provide normative rules for how a Client interacts with a Resource Server. This is because Clients may be provided by the Service or Content Provider who is running the Resource Server, so there is less need for an interoperable solution.

Nonetheless a pattern of interactions can be suggested which is well suited to most use cases.

(A) The Client issues an “HTTP Get” for a Protected Resource controlled by a Service Provider

(B) The Service Provider issues a “401” challenge, providing metadata with the challenge indicating that an Access Token is required. This metadata includes the OMAP "scope" string equivalent to the requested resource as well as a “recommended scope” string to use in an Authorization Request. It may also include a list of Authorization Server endpoints.

(C) The Client uses the “scope” string to determine if it has an Access Token in its token cache which supersets the “scope” required for this content. If not, it uses the “recommended scope” string in an Authorization Request to the AS with which the End-user has a relationship.

(D) Client does HTTP Get this time with the access token in the authorization header.