IMS Final Release

IMS Security Framework

IMS Final Release
Version 1.0

Date Issued:	15 May 2019
Status:	This document is made available for adoption by the public community at large.
This version:	https://www.imsglobal.org/spec/security/v1p0/
Latest version:	https://www.imsglobal.org/spec/security/latest/
Errata:	https://www.imsglobal.org/spec/security/v1p0/errata/

IPR and Distribution Notices

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

IMS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on IMS's procedures with respect to rights in IMS specifications can be found at the IMS Intellectual Property Rights web page: http://www.imsglobal.org/ipr/imsipr_policyFinal.pdf.

Use of this specification to develop products or services is governed by the license with IMS found on the IMS website: http://www.imsglobal.org/speclicense.html.

Permission is granted to all parties to use excerpts from this document as needed in producing requests for proposals.

The limited permissions granted above are perpetual and will not be revoked by IMS or its successors or assigns.

THIS SPECIFICATION IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NONINFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE CONSORTIUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS SPECIFICATION.

Public contributions, comments and questions can be posted here: http://www.imsglobal.org/forums/ims-glc-public-forums-and-resources.

→

Abstract

IMS Global has created, is creating, and will create, service-oriented and message-exchange interoperability specifications. These specifications recommend or require a number of different security patterns: for example, the use of OAuth 1.0 based message signing, OAuth 2 based authentication and authorization, and so forth. In this document, IMS defines a set of patterns for security that all of its specifications SHOULD use (only in special circumstances will IMS consider exceptions). These security patterns are based upon the appropriate standards and specifications published by other organizations: for example the Internet Engineering Task Force (IETF) and its Requests For Comments (RFCs).

This security framework has three basic patterns for adoption:

- Use of the OAuth 2.0 Client Credential Grant mechanism to secure web services between trusted systems (this MAY make use of JSON Web Tokens, JWT, for the access tokens);
- Use of the OAuth 2.0 Authorization Code Grant mechanism to secure web services between systems where there is no pre-established trust relationship (this MAY make use of JWT for the access tokens);
- Use of OpenID Connect with JWT-based message exchanges to secure browser-instigated exchanges between a tool and the launching platform.

All new IMS specifications MUST review the security patterns defined herein and adopt those that are suitable. IMS MAY revise specifications that have already been published to align with these security patterns.

Table of Contents

› 1. Introduction

› 1.1 Scope and Context

Adopters of IMS specifications will refer to this document with respect to the security approaches they MUST use. The aim is to require all of the IMS specifications to use a common security framework. Use of a common security framework promotes a consistent and compatible implementation requirement that simplifies adoption when more than one IMS specification is being implemented.

All IMS service-based specifications MUST make reference to this document. The IMS service specifications will cite appropriate sections of this document. In some cases, IMS specifications MAY contain exceptions to the recommendations made in this document. These exceptions

MUST be explained and justified in the IMS specification citing this document. The Best Practice and/or Implementation Guide documents for the relevant IMS specification MUST provide further explanation to implementers regarding the manner in which they are to implement security requirements in the context of that specification.

› 1.2 Terminology

This specification defines the following terms:

Authentication
Process used to achieve sufficient confidence in the binding between an entity and the presented Identity.

Claim
Piece of information asserted about an entity.

Consumer
An entity for which an end user gains access through a ***Platform***. The Consumer may make use of, or provide services to, the Platform.

ID Token
JWT that contains ***Claims*** about the ***Authentication*** event. It MAY contain other Claims.

Identifier
Value that uniquely characterizes an entity in a specific context.

Identity
Set of attributes related to an entity.

Issuer
Entity that issues a set of ***Claims***. The Issuer is the entity that starts an information exchange and as such could be either a Platform or a Consumer.

Issuer Identifier
Verifiable identifier for an ***Issuer***. An Issuer Identifier is a case-sensitive URL, using the HTTPS scheme, that contains scheme, host, and optionally, port number, and path components, and no query or fragment components.

Message
Request or a response between a ***Consumer*** and a ***Platform***.

Platform
An entity through which an end user interacts to gain access to some remotely launched ***Tool*** or a ***Consumer***. The Platform may make use of, or provide services to, the Tool or may provide services to the Consumer.

Private Key
The private key in the private-public key pair. This is the secret key that is used by the Issuer,

of a message, to sign the JWT using the JSON Web Signature [RFC7515].

Public Key
The public key in the private-public key pair. This is the key used at the receiving system to authenticate the message and confirm it has not been altered during transmission.

Relying Party
OAuth 2.0 Client application requiring End-User Authentication and Claims from an OpenID Provider.

Subject Identifier
Locally unique and never-reassigned identifier within the ***Issuer*** for the end user, which is intended to be consumed by the ***Consumer***. This identifier MUST be the same as the Issuer's User ID for that end user.

Tool
An entity which has been launched by a ***Platform***. The Tool may make use of, or provide services to, the Platform.

Validation
Process intended to establish the soundness or correctness of a construct in the context of the original requirement.

Verification
Process intended to test or prove the truth or accuracy of a fact or value.

Voluntary Claim
Claim specified by the ***Consumer*** as being useful but not essential for the specific task requested by the end user.

This specification also uses the terms:

"Client", "Client Authentication", "Client Identifier" and "Scope"
As defined by OAuth 2.0 [RFC6749].

"Claim List", "Claim Name", "Claim Value", "JSON Web Token (JWT)", "JWT Claims Set", "Nested JWT", and "String or URI"
As defined by JSON Web Token [RFC7519]

"Code Verifier", "Code Challenge" and "Code Challenge Method"
As defined by Proof Key for Code Exchange [RFC7636]

"Header Parameter" and "JOSE Header"
As defined by JSON Web Signature (JWS) [RFC7515]

"User Agent"
As defined by [RFC2616].

The terminology definitions in this section, are a normative portion of this specification, imposing requirements upon implementations. All the capitalized words in the text of this specification,

such as "Issuer Identifier", refer to these defined terms. Whenever readers encounter them, they must follow their definitions found in this section.

Note that the term "Client" is used as defined by OAuth 2.0. Therefore, communication being secured is between a Platform and a Consumer. Depending on the specific exchange choreography either of these could be the Client. From the perspective of a service either the Platform or the Consumer could initiate the communication exchange.

For more background on some of the terminology used, see Internet Security Glossary, Version 2 [RFC4949], ISO/IEC 29115 Entity Authentication Assurance [ISO29115], and ITU-T X.1252 [ITU-X1252].

› 1.3 Acronyms

CSRF
Cross-Site Request Forgery

HTTP
Hypertext Transport Protocol

HTTPS
Hypertext Transport Protocol Secure

IANA
Internet Assigned Number Authority

IETF
Internet Engineering Task Force

ISO
International Standards Organization

ITU
International Telecommunications Union

JOSE
JSON Object Signing and Encryption

JSON
Java Script Object Notation

JWA
JSON Web Algorithms

JWE
JSON Web Encryption

JWK
JSON Web Key

JWS
JSON Web Signature

JWT
JSON Web Token

LTI
Learning Tools Interoperability

MAC
Message Authentication Code

PKCE
Proof Key for Code Exchange

RFC
Request for Comments

SSL
Secure Sockets Layer

TLS
Transport Layer Security

URL
Uniform Resource Locator

› 1.4 Conformance Statements

All sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

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].

An implementation of this specification that fails to implement a MUST/REQUIRED/SHALL requirement or fails to abide by a MUST NOT/SHALL NOT prohibition is considered nonconformant. SHOULD/SHOULD NOT/RECOMMENDED statements constitute a best practice. Ignoring a best practice does not violate conformance but a decision to disregard such guidance should be carefully considered. MAY/OPTIONAL statements indicate that implementers are entirely free to choose whether or not to implement the option.

The Conformance and Certification Guide for this specification may introduce greater normative constraints than those defined here for specific service or implementation categories.

› 2. Security Architecture

› 2.1 Web Services-based Architectures

Some IMS specifications describe a set of web service calls that can occur between a service consumer (Consumer) and a service provider (Platform). Typically a service call occurs between a Consumer and Platform when one 'pulls' data from the other (using an HTTP GET) or 'pushes' data (using HTTP PUT or POST) to the other. Figure 1 is a schematic representation of how this security framework expects a Consumer and Platform to perform these service calls.

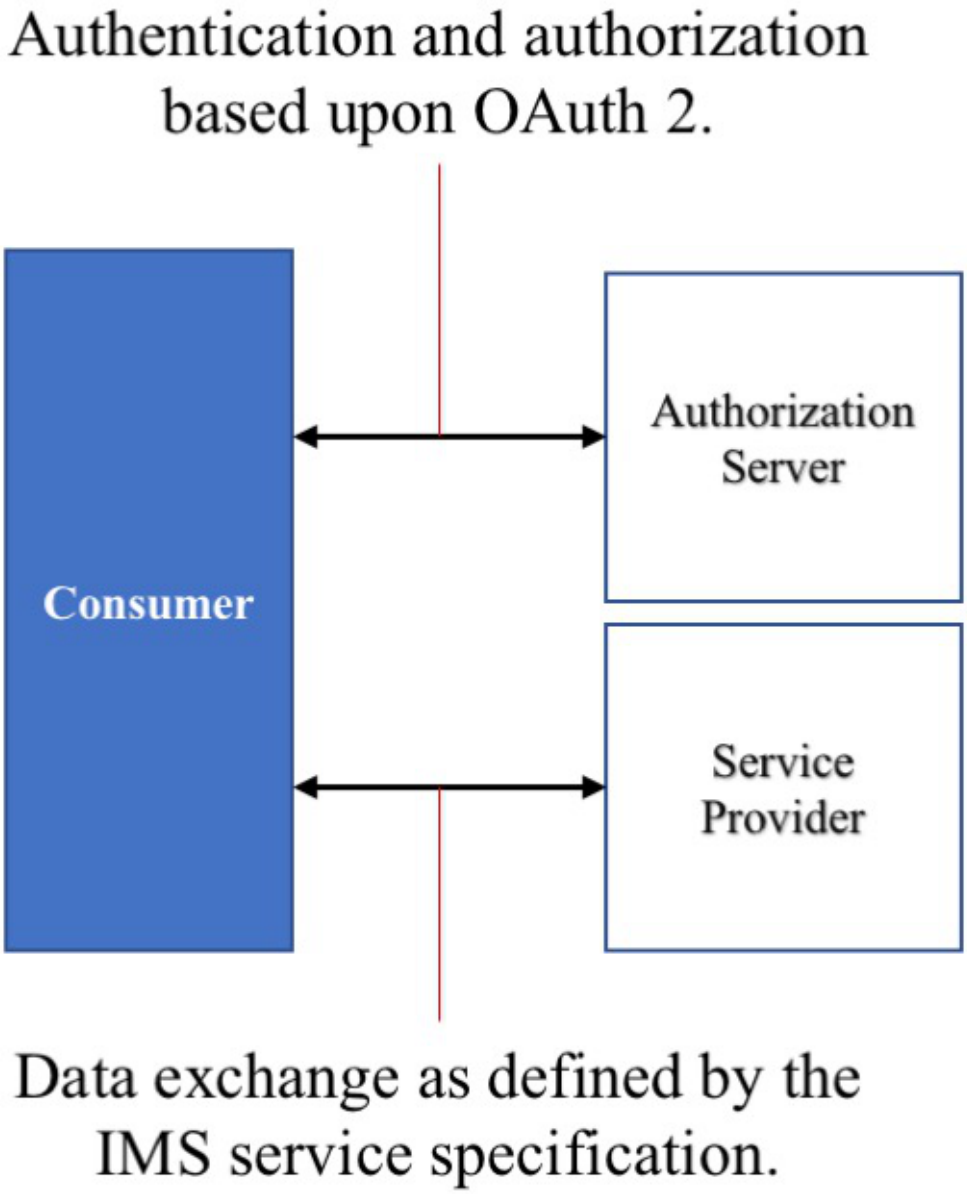

Figure 1 Web services architecture.

Each IMS specification will define how the Consumer and Platform will exchange information. When a service call MUST be done securely, the sender MUST use the methods described in this framework to secure the data transfer. This document defines how to achieve the "Authentication and Authorization" portion of the schema represented in Figure 1 and how each data exchange will use this information. The "Authentication and Authorization" portion of the exchange is based on the OAuth 2.0 protocol flow (as defined in [RFC6749] section 1.2). Sequences 'A' - 'D' in Figure 2 shows the abstract protocol flow for acquiring the needed authorization information, and sequences 'E' - 'F' show the flow of using the authorization information to perform the service call. Note that in Figure 2, the Client, as defined by OAuth 2.0, could be either the Platform or Consumer depending on which entity initiates the exchange.

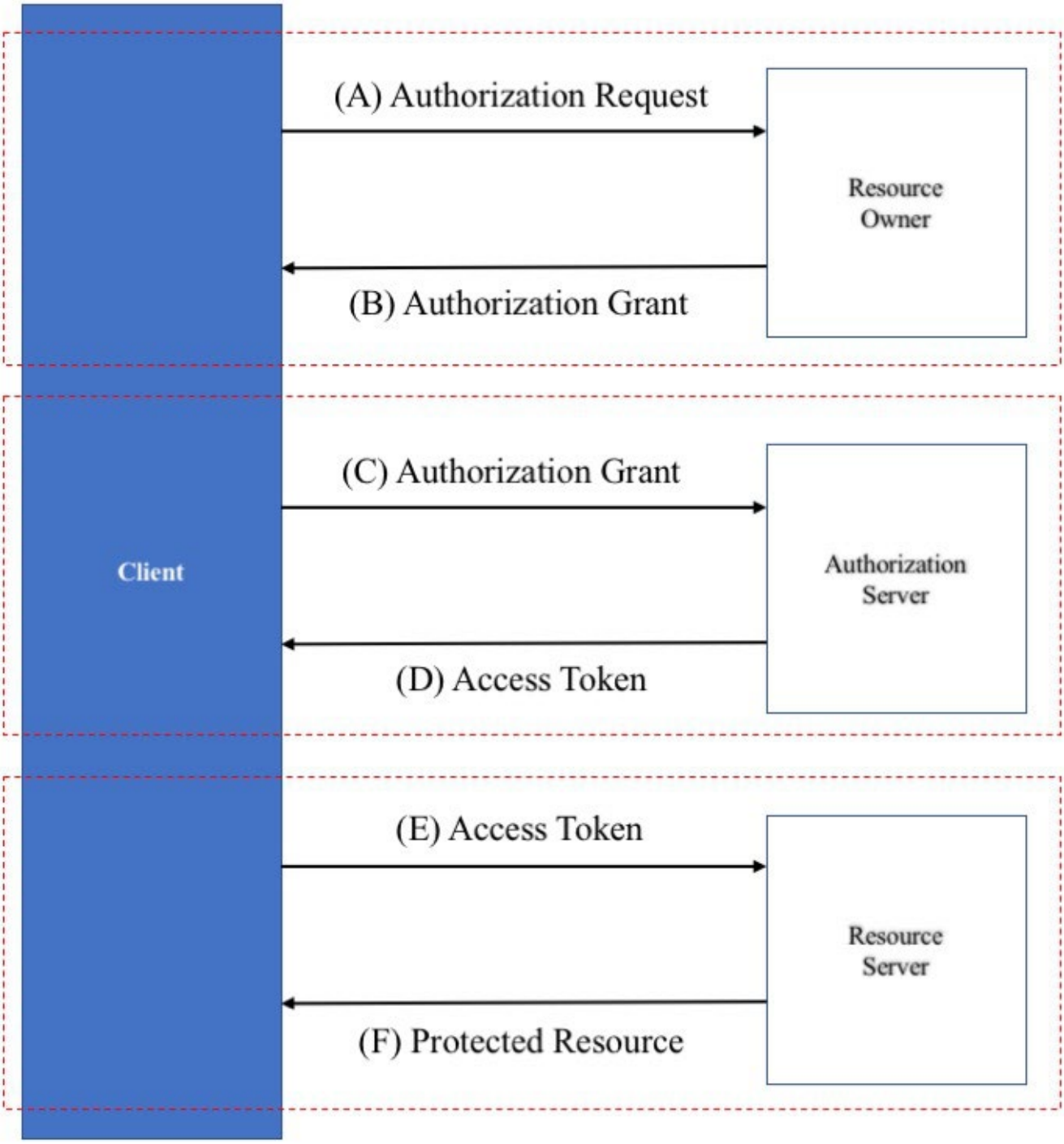

Figure 2 Web services abstract protocol flow.

In OAuth 2.0 there are four functional components in the authentication and authorization flow:

Client. The system, application, or tool that requires access to the corresponding resource via a specific endpoint or set of endpoints.

Resource Owner. The system, application, or tool that owns the resource located at a specific endpoint or set of endpoints and which can provide permission to obtain access to that resource.

Authorization Server. The system responsible for allocating the appropriate access authorization

using the authentication information supplied to it by the Client.

Resource Server. The system, application, or tool that makes the corresponding resource available via the specific endpoint or set of endpoints and that supports the defined authorization mechanism.

In general, a Client may obtain authorization through three phases:

1. Obtain authorization to resource (Phase 1) - denoted by sequence 'A' and 'B' in Figure 2.
2. Obtain the access information from the authorization server (Phase 2) - denoted by sequence 'C' and 'D' in the Figure 2. If a system does not support Phase 1, the required permissions information SHOULD be obtained via some out-of-band process.
3. Obtain access to the resource (Phase 3) - denoted by sequence 'E' and 'F' in the figure. If a system does not support Phase 2, then the required access information SHOULD be obtained via some out-of-band process.

In cases where there is an established trust between the Clients and Platforms, only Phases 2 and 3 will be used, relying on shared credentials obtained out-of-band (see OAuth 2.0 [RFC6749] Section 4.4). In cases where there is no pre-established trust relationship all three Phases MUST be used.

› 2.2 Non-Web Services-based Architectures

In the case where IMS has defined a non-web services based standard, the specification will describe the set of messages that can occur between a Platform and a Tool. In scenarios where the message exchange is vulnerable (for example, when launching from a web browser), the messages will be signed. This signing MAY include data derived from the identity-based authentication. For IMS specifications using a non-web services approach, Figure 3 shows a schematic representation of this security framework.

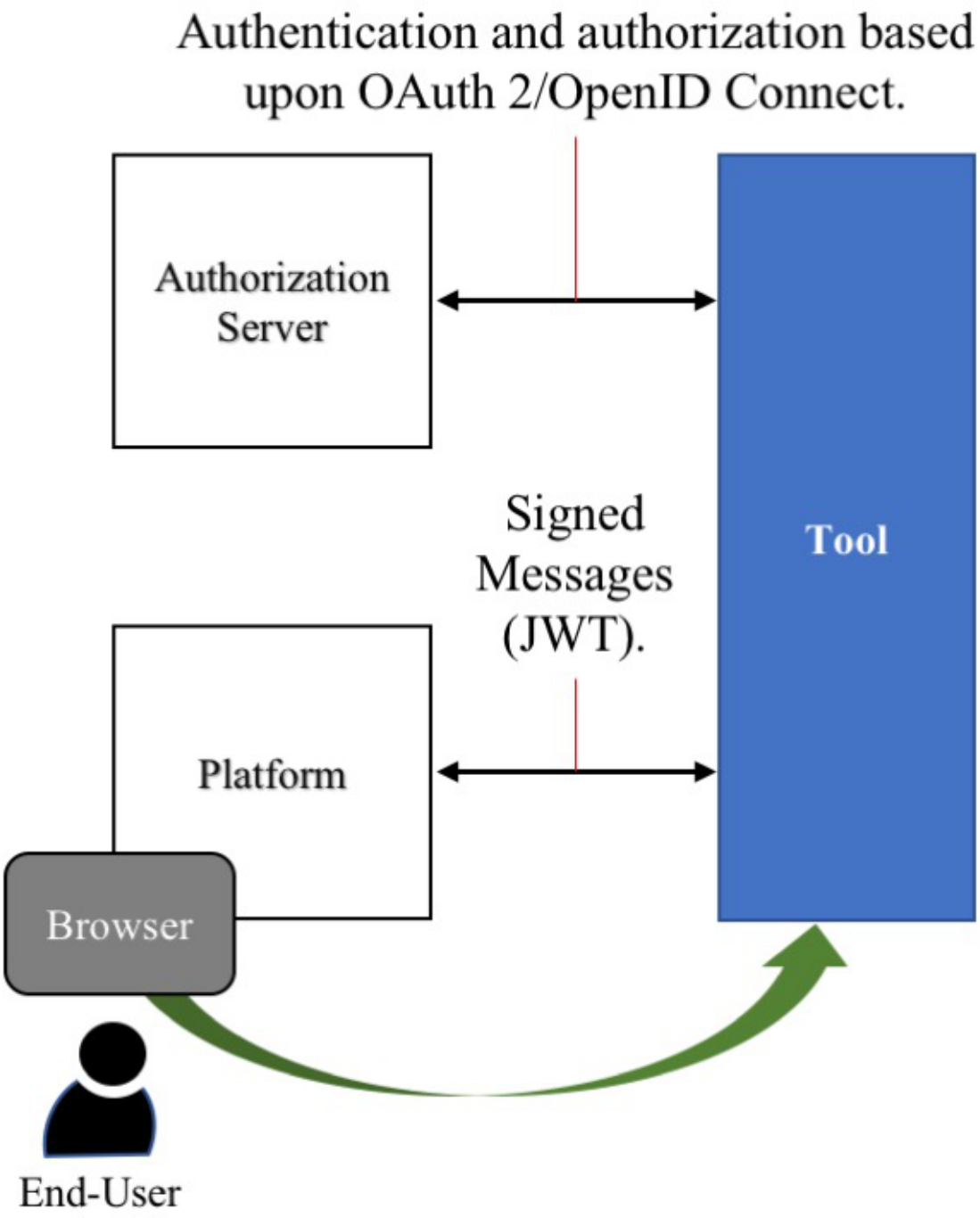

Figure 3 Non-web services architecture.

The IMS specification defines how a Tool can transform the messages exchanged between the Platform and the Tool (including a user's browser-based interaction) into a Tool-based experience. This document defines how to achieve Authentication and Authorization using a separate set of message exchanges between Platform and Tool and how to encode the authorization and authentication information in JWT-based message signing of these message exchanges. The authorization and authentication process uses an authorization server which MAY be a system independent of the Platform or MAY be endpoints hosted by the Platform.

› 3. Transport Security

Senders and receivers of data should encrypt the data to ensure that third parties cannot read the data in transit (for example, by sniffing packets).

Therefore Platforms and Consumers MUST send all requests and responses using Transport Layer Security (TLS). Exchange of the signed certificates for endpoints between Platforms and Consumers is beyond the scope of this specification. IMS advises implementers to refer to the various third-party certificate-signing services in order to obtain signed certificates.

Implementers MUST use TLS 1.2 (see [RFC5246]) and/or TLS 1.3 (see [RFC8446]). Note that [RFC8446] obsoletes [RFC5246], therefore it is RECOMMENDED that [RFC8446] is used. Implementers MUST NOT use Secure Sockets Layer (SSL).

› 4. Securing Web Services

› 4.1 Using OAuth 2.0 Client-Credentials Grant

When there is an established trust relationship between the resource owner, the resource server and the system requiring access to the resource(s), the approach shown in Figure 4 MUST be used for the web services. It is assumed that the OAuth 2 Client is the Consumer and the Resource Server the Platform. Authentication and Authorization requires the use of OAuth 2.0 bearer tokens, obtained using the mechanism described in Section 4.4 of [RFC6749].

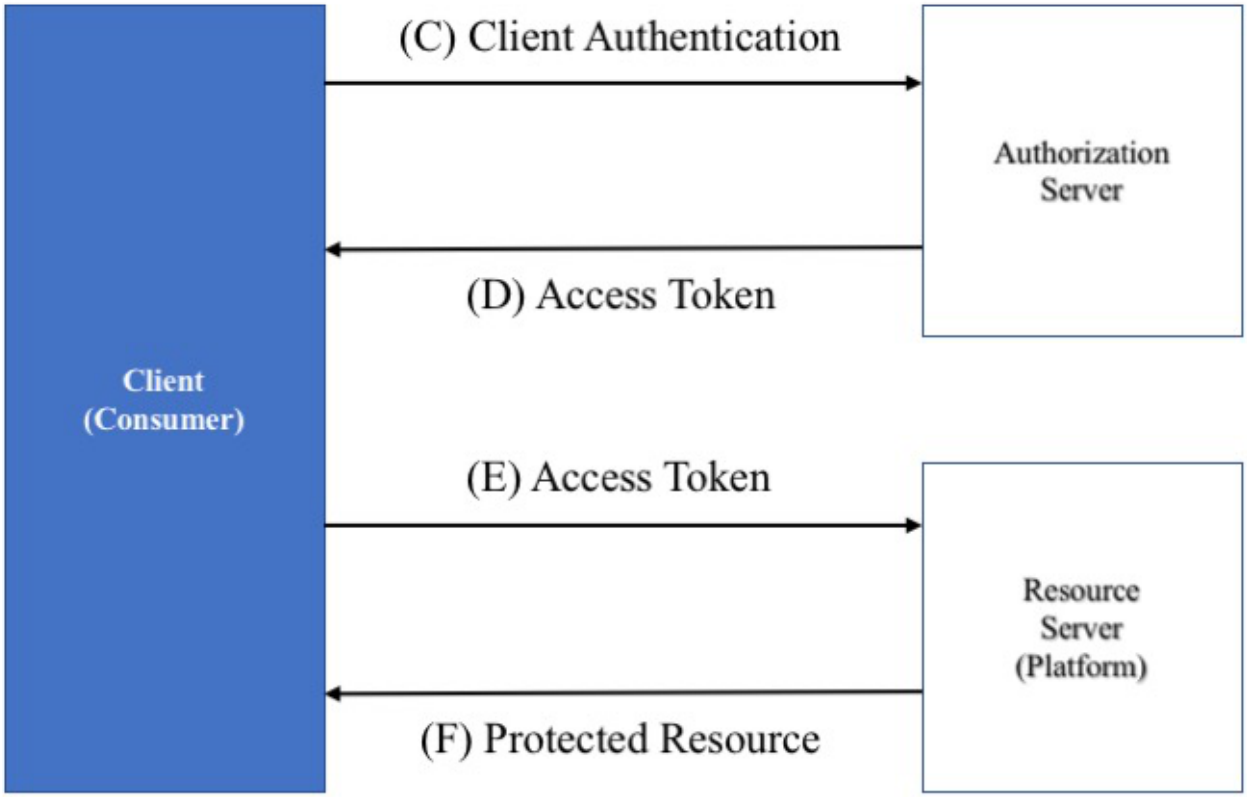

Figure 4 OAuth 2 client credentials based resource access.

Making a secured web service request comprises two steps:

1. Obtain an access token by sending a request to the Platform's OAuth 2 Access Token service endpoint (as in flow 'C/D' in the figure);
2. Include the access token in the `Authorization` header of service request messages (as in flow 'E/F' in the figure).

Once obtained, the Consumer can freely re-use the access token up until the token's expiry time, so that the Consumer need not repeat step 'C' for every service request. The specific, appropriate mechanism for obtaining the set of credentials is not within the scope of this document.

In IMS service specifications, scopes, as defined in [RFC6749], MUST be used when requesting an access token. The authorization server is responsible for validating the scopes identified in the request and the response MUST include a scope parameter which confirms this list or comprises a subset of the services requested. The Authorization Server may refuse to provide an access token and so MUST return an error response.

Consumers MUST use the OAuth 2.0 Client Credentials grant type. In this grant type, the Consumer can request an access token using only its credentials (using its consumer key and secret information) in these cases:

- The Consumer is requesting access to the protected resources under its control;
- The Consumer is requesting access to resources owned by another resource owner that has previously arranged this access permission with the authorization server.

In this approach, the Consumer issues a Consumer authentication request and receives an access token in response. Section 5 of [RFC6749] defines the issuing of an access token. To obtain an access token for IMS service access requires four pieces of configuration information:

Key. The public identifier for the communication exchange (this is also referred to as the application ID).

Secret. The shared secret for the communication exchange (this is also referred to as the application secret).

List of Scopes. The list of scopes that identify the set of endpoints for which access permission is being requested.

OAuth 2 Access Token Service Endpoint. The endpoint from which the approved, requesting Consumer can obtain the access token.

Requests for an access token use an HTTP POST and TLS. The Consumer MUST use its key and secret with the HTTP Basic Authentication method (as described in [RFC2617]) for this request and MUST NOT put its key, secret and list of scopes into the request body.

For client credentials based IMS Web Services, the set of query parameters (Section 4.4 of [RFC6749]) used to request an access token are:

grant_type
REQUIRED. Value MUST be set to "client_credentials".

scope
REQUIRED. The scope of the access request.

An example of the request message is shown below in which three scopes (scopename1, scopename2 and scopenamex) are identified:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&scope=scopename1%20scopename2%20scopenamex
```

The recommended naming convention to be used for scopes is described in Scope Naming Conventions of this document.

If the authorization service successfully grants this request (see Section 5.1 in [RFC6749] for the detailed description), it responds with an HTTP 200 OK response containing the access token and its expiry lifetime (IMS recommends a default expiry lifetime of 3600 seconds, one hour, for access tokens) and confirms the set of scopes supported by this access token:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token" : "2YotnFZFEjr1zCsicMWpAA",
    "stenotype" : "bearer",
    "expires_in" : 3600,
    "scope" : "scopename1 scopename2 scopenamex"
}
```

The Consumer utilizes the access token to authenticate with the resource using the HTTP Authorization request header field [RFC2617] with an authentication scheme defined by the specification of the access token type used, such as [RFC6750]. For example, with a bearer-type token, a resource retrieval request uses this form:

```
GET /resource/1 HTTP/1.1
Host: provider.example.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
```

The authorization server MAY decide not to issue an access token: this could be because the request scopes are invalid, the credentials from the client may be invalid, etc. In this case the authorization server MUST return an error message (see Section 5.2 in [RFC6749] for the detailed description) with an HTTP 400 (Bad Request) status code. An example of the error response message (when the requested scope is invalid) is:

```
HTTP/1.1 400 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "error" : "invalid_scope"
}
```

› 4.1.1 Using JSON Web Tokens with OAuth 2.0 Client-Credentials Grant

In this approach, a Consumer generates a JWT [RFC7519] bearer token as a means for authentication in requesting an OAuth 2.0 access token, which it can then use to authorize requests to services as per [RFC7523].

The JWT payload MUST contain at least the following Claims:

`iss`	A unique identifier for the entity that issued the JWT
`sub`	"client_id" of the OAuth Consumer
`aud`	Authorization server identifier (s)
`iat`	Timestamp for when the JWT was created
`exp`	Timestamp for when the JWT should be treated as having expired (after allowing a margin for clock skew)
`jti`	A unique (potentially reusable) identifier for the token

```
{
    "iss" : "tool.com",
    "sub" : "www.example.com",
    "aud" : ["https://www.example.com/lti/auth/token"],
    "iat" : "1485907200",
    "exp" : "1485907500",
    "jti" : "29f90c047a44b2ece73d00a09364d49b"
}
```

The `aud` Claim MUST contain a value that identifies the authorization server as an intended audience. The precise strings to be used as the audience for a given authorization server MUST be configured out of band by the authorization server and the issuer of the JWT. For example, the authorization server MAY instruct the Consumer to use the token endpoint URL of the authorization server as a value for an `aud` element to identify the authorization server as an intended audience of the JWT. The authorization server MUST reject any JWT that does not contain its own identity as the intended audience.

The `iat` Claim MUST be the time at which the Consumer generated the JWT.

The `exp` Claim MUST be an absolute expiry time for the message (typically five minutes after the `iat` timestamp); the Consumer MUST honor this expiry time, though it MAY also choose to expire

the JWT at an earlier time (but no earlier than the `iat` value). This `exp` timestamp SHOULD also indicate when the `jti` value expires and could be re-used.

The Consumer MAY include other top-level Claims in the JWT and, if it does so, the authorization server MAY ignore them.

The above JSON structure represents the first, payload element of the JWT. The Consumer MUST also add a second, header element, referred to as the JSON Object Signing and Encryption (JOSE) header, that defines the algorithm it used to sign the token.

```
{
    "typ" : "JWT",
    "alg" : "RS256"
}
```

The Consumer MUST generate a third, signature element by applying the algorithm to the contents of both the header and payload elements (see [RFC7515] for how to create a JSON Web Signature, JWS). The Consumer then forms the value of the JWT by separately base-64 encoding each of the three elements and concatenating them with a period (`.`) as a separator character i.e.

```
JWT = base64_encode(JOSE Header).base64_encode(Claim Set
Payload).base64_encode(JWS Signature)
```

creating a JWT of (line breaks added for clarity):

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiJ0b29sLmNvbSIsInN1YiI6Ind3dy5leGFtcGxlLmNvbSIsImF1ZCI6Imh0dHBzOi8vd3d
3Lm
V4YW1wbGUuY29tL2x0aS9hdXRoL3Rva2VuIiwiaWF0IjoiMTQ4NTkwNzIwMCIsImV4cCI6IjE0ODU5MD
c1
MDAiLCJqdGkiOiIyOWY5MGMwNDdhNDRiMmVjZTczZDAwYTA5MzY0ZDQ5YiJ9
.liArqLDIF-xGcCu8ythy0HlzntxwZ90AYTnwH-daCQQ
```

The Consumer's resulting authorization request uses the following POST parameters:

- `grant_type`: `client_credentials`
- `client_assertion_type`: `urn:ietf:params:oauth:client-assertion-type:jwt-bearer`
- `client_assertion`: the Consumer's generated JWT
- `scope`: `https://purl.imsglobal.org/spec/lti-ags/scope/lineitem`
  `https://purl.imsglobal.org/spec/lti-ags/scope/result/read`

This leads to a request with the following form (with line breaks for clarity):

```
POST /lti/auth/token HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%
3Aclient-assertion-type%3Ajwt-bearer
&client_assertion=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiJ0b29sLmNvbSIsInN1YiI6Ind3dy5leGFtcGxlLmNvbSIsImF1ZCI6Imh0dHBzOi8vd3d
3LmV4YW1wbGUuY29tL2x0aS9hdXRoL3R
va2VuIiwiaWF0IjoiMTQ4NTkwNzIwMCIsImV4cCI6IjE0ODU5MDc1MDAiLCJqdGkiOiIyOWY5MGMwNDd
hNDRiMmVjZTczZDAwYTA5MzY0ZDQ5YiJ9
.liArqLDIF-xGcCu8ythy0HlzntxwZ90AYTnwH-daCQQ
&scope=http%3A%2F%2Fimsglobal.org%2Fspec%2Flti-
ags%2Fscope%2Flineitem%20http%3A%2F%2Fimsglobal.org%2Fspec%2Flti-
ags%2Fscope%2Fresult%2Fread
```

The authorization server decodes the JWT and MUST validate the values for the `iss`, `sub`, `exp`, `aud` and `jti` claims, followed by verifying the signature. If it finds the request to be a valid, it generates and returns an access token, with a response of this form:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token" : "dkj4985kjaIAJDJ89kl8rkn5",
    "token_type" : "bearer",
    "expires_in" : 3600,
    "scope" : "https://purl.imsglobal.org/spec/lti-ags/scope/lineitem
https://purl.imsglobal.org/spec/lti-ags/scope/result/read"
}
```

As per [RFC7519] the token MUST ONLY contain ASCII characters (0x20-0x7A) and SHOULD be at least 15 characters in length. The token MAY be a JWT (see below). IMS recommends a default expiry time of 1 hour (3600 seconds). The bearer of the token can use it until the token expires (as evidenced by a failed request) but IMS recommends that a Consumer manages tokens such that once they have expired (allowing for clock skew between the systems), it requests a new one when needed.

› *4.1.1.1 Using a JWT as an Access Token*

Consumers SHOULD treat the access token returned by the authorization server as an opaque

string. Any meaning that the token may have is relevant only to the authorization server and resource provider. If a resource provider does not wish to manage access tokens, it can use a JWT to encapsulate the details of the token so that the bearer provides the details with each service request for verification.

Following from the above example, this might be the access JWT token's payload:

```
{
    "sub" : "www.example.com",
    "iat" : "1485907200",
    "exp" : "1485907500",
    "imsglobal.org.security.scope" : "https://purl.imsglobal.org/spec/lti-
ags/scope/lineitem https://purl.imsglobal.org/spec/lti-ags/scope/result/read"
}
```

This might be the JWT's header:

```
{
    "typ" : "JWT",
    "alg" : "RS256"
}
```

After signing, this would create the JWT:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3d3cuZXhhbXBsZS5jb20iLCJpYXQiOiI
xNDg1OTA3MjAwIiwiZXhwIjoiMTQ4NTkwNzUwMCIsImltc2dsb2JhbC5vcmcuc2VjdXJpdHkuc2NvcGU
iOiJMdGlMaW5rU2V0dGluZ3MgU2NvcmUuaXRlbS5QVVQifQ.UWCuoD05KDYVQHEcciTV88YYtWWMwgb3
sTbrjwxGBZA
```

Thus, the authorization server might return this access token in a response like this:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token" :
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3d3cuZXhhbXBsZS5jb20iLCJpYXQiOi
IxNDg1OTA3MjAwIiwiZXhwIjoiMTQ4NTkwNzUwMCIsImltc2dsb2JhbC5vcmcuc2VjdXJpdHkuc2NvcG
UiOiJMdGlMaW5rU2V0dGluZ3MgU2NvcmUuaXRlbS5QVVQifQ.UWCuoD05KDYVQHEcciTV88YYtWWMwgb
3sTbrjwxGBZA",
    "token_type" : "bearer",
    "expires_in" : 3600,
    "scope" : "https://purl.imsglobal.org/spec/lti-ags/scope/lineitem
```

```
    https://purl.imsglobal.org/spec/lti-ags/scope/result/read"
  }
```

When the resource provider receives a service request with this access token, it can verify the signature and extract the details of the validity of the request from the JWT before proceeding to process the request.

› 4.2 Using OAuth 2.0 Authorization Code Grant

In scenarios where the learner or another third party is the resource owner, there will not be a pre-established trust relationship. Therefore the client credentials approach is insufficient. Instead IMS RECOMMEND the use of OAuth 2.0 Authorization Code Grant. Figure 5 shows the assumed system architecture when the resource owner is required to provide explicit permission for access to a resource. It is assumed that the OAuth 2 Client is the Consumer and the Resource Server the Platform. The User Agent could be a Browser. Authentication and Authorization requires the use of OAuth 2.0 bearer tokens, obtained using the mechanism described in Section 4.1 of [RFC6749].

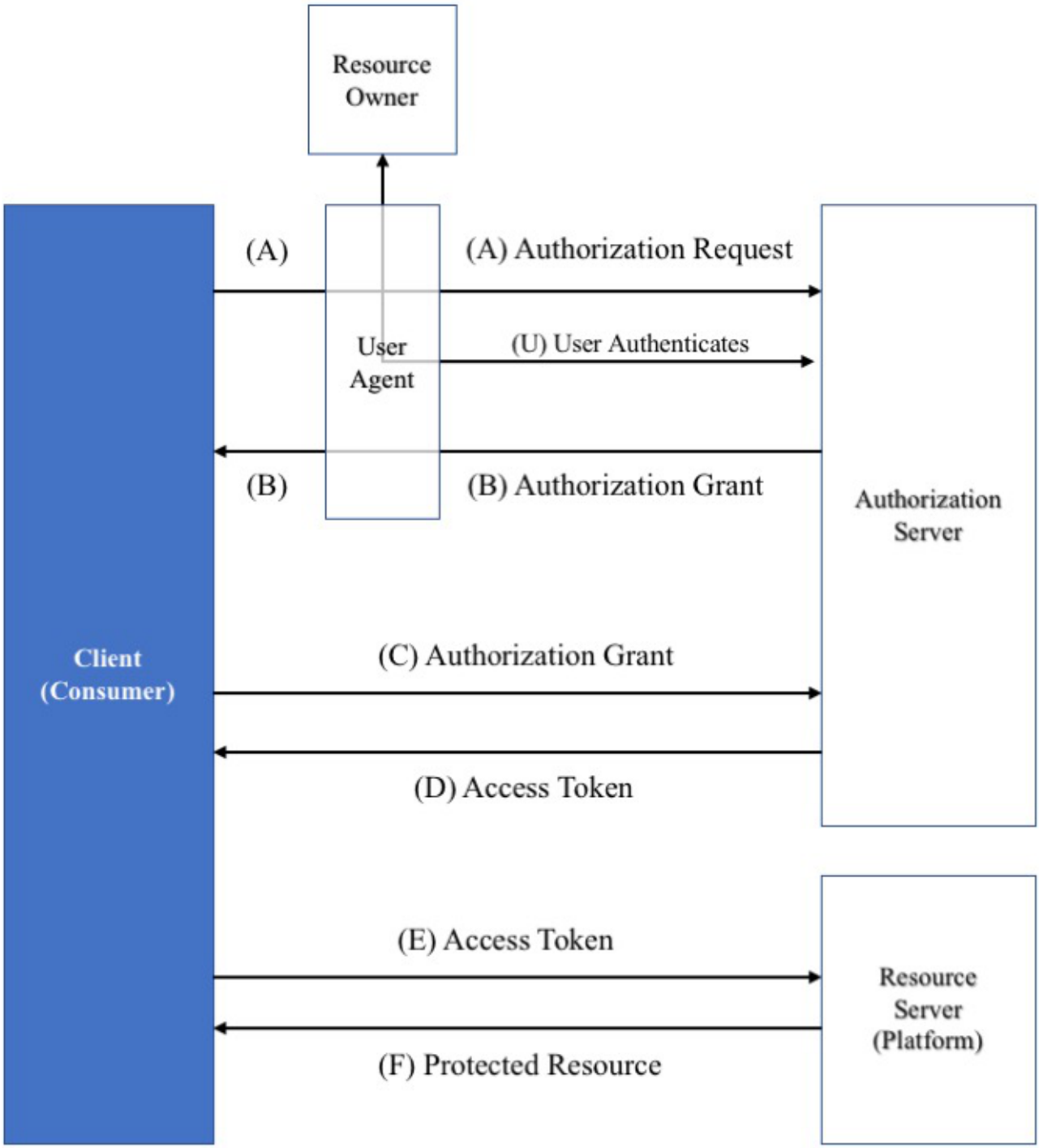

Figure 5 OAuth 2 authorization code grant based resource access.

Making a secured web service request using authorization code grant comprises three steps:

1. Obtain an authorization code using a choreography between the Consumer, User Agent, Resource Owner and Authorization Server (as in flow A/U/B in the figure);
2. Obtain an access token by sending a request, using the previously obtained authorization code, to the Platform's OAuth 2 Access Token service endpoint (as in flow 'C/D' in the figure);
3. Include the access token in the `Authorization` header of service request messages (as in

flow 'E/F' in the figure).

Once obtained, the Consumer can freely re-use the access token up until the token's expiry time, so that the Consumer need not repeat step 'C' for every service request. Token refresh is also available and if adopted MUST be used as described in [RFC6749] Section 5.2

In IMS service specifications, the 'state' parameter, as defined in [RFC6749], MUST be used when requesting an access token. The state parameter is 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. This parameter MUST be used for preventing cross-site request forgery.

In IMS service specifications, the 'scope' parameter, as defined in [RFC6749], MUST be used when requesting an access token. The authorization server is responsible for validating the scopes identified in the request and the response MUST include a scope parameter which confirms this list or comprises a subset of the services requested. The Authorization Server may refuse to provide an access token and so MUST return an error response.

In the OAuth 2.0 Security Best Practices document [OAUTH2-SBP] the use of Proof Key for Code Exchange (PKCE) [RFC7636] is recommended in order to (with the help of the authorization server) detect and prevent attempts to inject (replay) authorization codes into the authorization response. When using IMS specifications, PKCE MUST be used to protect Authorization Code Grant based access. PKCE requires the Consumer to create a code verifier which is encoded and supplied as a code challenge. The authorization server MUST return the code verifier and the Consumer checks the validity of this returned value.

For authorization code grant based IMS Web Services, the set of query parameters (Section 4.1 of [RFC6749]) used to request an authorization code are:

resonse_type
REQUIRED. Value MUST be set to "code".

client_id
REQUIRED. The client identifier.

redirect_uri
OPTIONAL.

scope
REQUIRED. The scope of the access request.

state
REQUIRED. An opaque value used by the client to maintain state between the request and callback.

code_challenge
REQUIRED. This is BASE64URL-ENCODE(SHA256(ASCII(code_verifier))).

code_challenge_method
REQUIRED. This MUST have a value of S256 i.e. the SHA256 code verifier transformation method is used.

The Consumer initiates the flow by directing the resource owner's user-agent to the authorization endpoint (flow (A) in Figure 5). All exchanges MUST use TLS. The Consumer 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). For IMS, the query parameters 'scope', 'state', 'code_challenge' and 'code_challenge_method' MUST be used ('code_challenge' and 'code_challenge_method' are required as part of the PKCE). An example of the request is shown below in which one scope (scopename1) is identified:

```
      GET /authorize?
response_type=code&client_id=s6BhdRkqt3&state=xyzjklabc&scope=scopename1&code_ch
allenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM

&code_challange_method=S256&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2F
cb HTTP/1.1
      Host: server.example.com
```

If the resource owner grants the access request, the authorization server issues an authorization code and delivers it to the Consumer by adding the specific parameters to the query component of the redirection URI (flow (B) in Figure 5. For example, the authorization server redirects the user-agent by sending the following HTTP response

```
      HTTP/1.1 302 Found
      Location: https://client.example.com/cb?
code=SplxlOBeZQQYbYS6WxSbIA&state=xyzjklabc
```

This authorization code MUST be used only once. A lifetime for the authorization code of 600 seconds (10 minutes) is RECOMMENDED. If an authorization code is used more than once, the authorization server MUST deny the request and SHOULD revoke (when possible) all tokens previously issued based on that authorization code. The authorization code is bound to the client identifier and redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the query component of the redirection URI. For example:

```
      HTTP/1.1 302 Found
      Location: https://client.example.com/cb?
error=access_denied&state=xyzjklabc
```

The next step is use the authorization code to obtain the access token (flow (C) in Figure 5). Requests for an access token use an HTTP POST and TLS. The Consumer MUST use its key and secret with the HTTP Basic Authentication method (as described in [RFC2617]) for this request and MUST NOT put its key and secret into the request body. As REQUIRED by PKCE, the code verifier query parameter MUST be supplied. Therefore for authorization code grant based IMS Web Services, the set of query parameters (Section 4.1 of [RFC6749]) used to request an access token are

grant_type
REQUIRED. Value MUST be set to "authorization_code".

code
REQUIRED. The authorization code received from the authorization server.

client_id
REQUIRED, if the client is not authenticating with the authorization server

redirect_uri
REQUIRED, if the "redirect_uri" parameter was included in the authorization request

scope
REQUIRED. The scope of the access request.

code_verifier
REQUIRED. Code verifier.

An example of the request message is shown below:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA&scope=scopename1&redir
ect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb&code_verifier=dBjftJeZ4CVP-
mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

If the authorization service successfully grants this request (see Section 5.1 in [RFC6749] for the detailed description), it responds with an HTTP 200 OK response (flow (D) in Figure 5 containing the access token and its expiry lifetime (IMS recommends a default expiry lifetime of 3600 seconds, one hour, for access tokens) and confirms the scopes:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
```

```
{
    "access_token" : "2YotnFZFEjr1zCsicMWpAA",
    "token_type" : "bearer",
    "expires_in" : 3600,
    "scope" : scopename1,
    "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

The Consumer utilizes the access token to authenticate with the resource using the HTTP Authorization request header field [RFC2617] with an authentication scheme defined by the specification of the access token type used, such as [RFC6750]. For example, with a bearer-type token, a resource retrieval request uses this form (flow (E) in Figure 5:

```
GET /resource/1 HTTP/1.1
Host: provider.example.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
```

The authorization server MAY decide not to issue an access token: this could be because the request scopes are invalid, the credentials from the client may be invalid, the PKCE code verifier value is invalid, etc. In this case the authorization server MUST return an error message (see Section 5.2 in [RFC6749] for the detailed description) with an HTTP 400 (Bad Request) status code. An example of the error response message (when the requested scope is invalid) is:

```
HTTP/1.1 400 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "error" : "invalid_scope"
}
```

› 4.2.1 Using JSON Web Tokens with OAuth 2.0 Authorization Code Grant

In this approach, a Consumer generates a JWT [RFC7519] bearer token as a means for authentication in requesting an OAuth 2.0 access token, which it can then use to authorize requests to services as per [RFC7523]. This is similar to the approach for using JSON Web Tokens with OAuth 2.0 Client-Credentials Grant (see Section 4.1). The JWT payload MUST use the same set of claims as list in Section 4.1. When using the same set of values as per Section 4.1 this leads to a request with the following form (with line breaks for clarity):

```
POST /lti/auth/token HTTP/1.1
Host: www.example.com
```

```
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=n0esc3NRze7LTCu7iYzS6a5acc3f0ogp4&client_asse
rtion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
&client_assertion=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiJ0b29sLmNvbSIsInN1YiI6Ind3dy5leGFtcGxlLmNvbSIsImF1ZCI6Imh0dHBzOi8vd3d
3LmV4YW1wbGUuY29tL2x0aS9hdXRoL3R
va2VuIiwiaWF0IjoiMTQ4NTkwNzIwMCIsImV4cCI6IjE0ODU5MDc1MDAiLCJqdGkiOiIyOWY5MGMwNDd
hNDRiMmVjZTczZDAwYTA5MzY0ZDQ5YiJ9
.liArqLDIF-xGcCu8ythy0HlzntxwZ90AYTnwH-daCQQ
&scope=http%3A%2F%2Fimsglobal.org%2Fspec%2Flti-
ags%2Fscope%2Flineitem%20http%3A%2F%2Fimsglobal.org%2Fspec%2Flti-
ags%2Fscope%2Fresult%2Fread
```

The authorization server decodes the JWT and MUST validate the values for the `iss`, `sub`, `exp`, `aud` and `jti` claims, followed by verifying the signature. If it finds the request to be a valid, it generates and returns an access token, with a response of this form:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token" : "dkj4985kjaIAJDJ89kl8rkn5",
    "token_type" : "bearer",
    "expires_in" : 3600,
    "scope" : "https://purl.imsglobal.org/spec/lti-ags/scope/lineitem
https://purl.imsglobal.org/spec/lti-ags/scope/result/read"
}
```

As per [RFC7519] the token MUST ONLY contain ASCII characters (0x20-0x7A) and SHOULD be at least 15 characters in length. The token MAY be a JWT (see below). IMS recommends a default expiry time of 1 hour (3600 seconds). The bearer of the token can use it until the token expires (as evidenced by a failed request) but IMS recommends that a Consumer manages tokens such that once they have expired (allowing for clock skew between the systems), it requests a new one when needed.

› *4.2.1.1 Using a JWT as an Access Token*

As in the case of Client Credentials, the access token itself could be a JWT. Therefore Consumers SHOULD treat the access token returned by the authorization server as an opaque string. Any meaning that the token may have is relevant only to the authorization server and resource provider. If a resource provider does not wish to manage access tokens, it can use a JWT to encapsulate the details of the token so that the bearer provides the details with each service

request for verification.

› 5. Message Security and Message Signing

When transferring a User Agent from one entity to another the redirecting party SHOULD sign the Message effecting this transfer using a JWT. Messages are JWTs sent through the browser by auto-POSTing from sender to some endpoint URL on the message receiver. The signature on the JWT is always signed by the private key of the message sender. Therefore, for platform-to-tool messages, the platform signs the JWT with their private key, and the client then verifies the signature on the message payload they receive by using the platform's public key they obtained out-of-band. In an example out-of-band registration workflow, the public-key/key-set-URL exchange can happen during Tool registration: the Platform registers the Consumer, gives the Consumer its public-key/key-set-URL; the Consumer reciprocates.

A Message MAY carry additional Claims related to the user interface flow presented to the end user: these Claims MUST be defined in the corresponding specification (for example, the LTI Specification [LTI-13], which defines the concepts of LTI-compliant platforms and tools).

› 5.1 Platform-Originating Messages

When a Platform acts as an Identity Provider (IDP), it MUST pass identity information with each Message. To do so, it MUST use a subset of Section 2 of OpenID Connect Core [OPENID-CCORE] (see Section 5.1.1 of that document for more details) to provide user Authentication to the Tool (see Figure 6 for a schematic representation of this architecture). This is done using a combination of the 'Initiating Login from a Third Party' and 'Implicit Flow' of the OpenID Connect Core [OPENID-CCORE] (Sections 4 and 3.2.1 respectively). The authentication relies upon the Platform and Tool being aware of various identifiers for each other as well as using public key encryption for signing the messages.

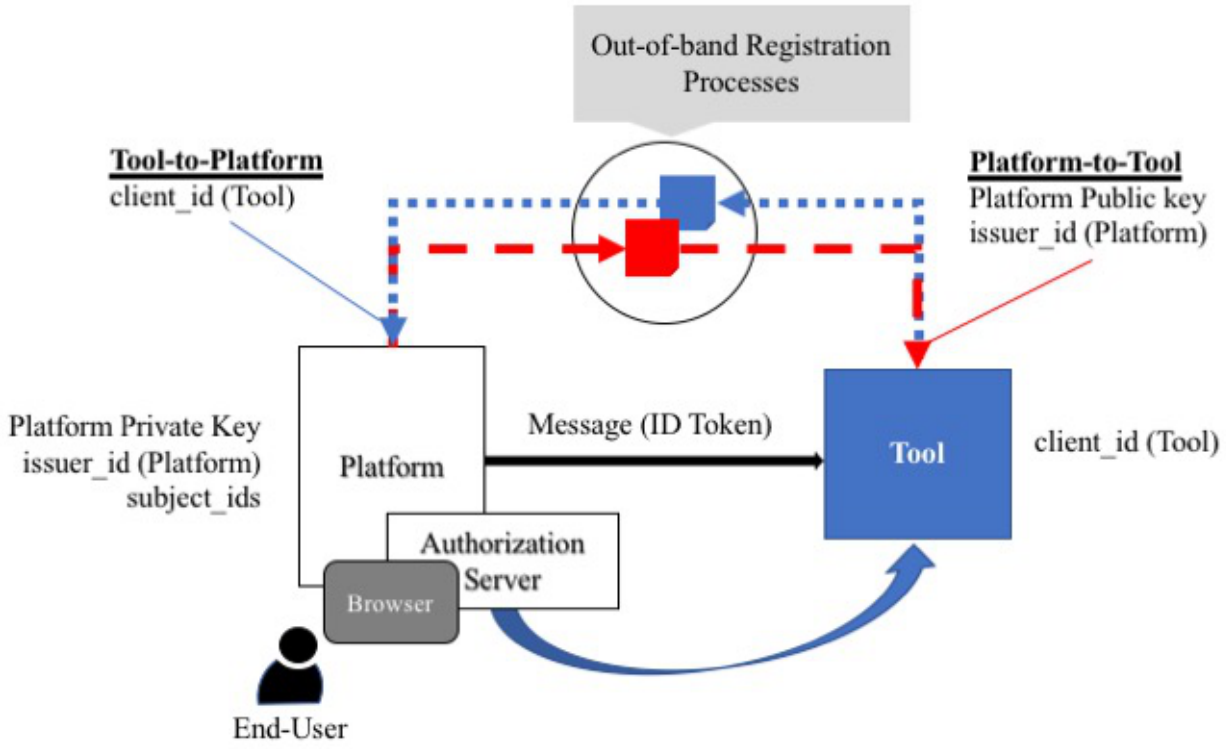

Figure 6 Information flow for platform-originating messages.

The key points in the figure above are:

- The Tool must have been allocated an OAuth 2 client_id (usually by an Authorization Server and this may be owned by the Platform);
- The Platform must be given the client_id for the Tool (using some out-of-band registration process if not assigned by the Platform);
- The Tool must be given the Public Key and Issuer Identifier (issuer_id) for the Platform (using some out-of-band registration process). See 6. Key Management for more details on Key Management;
- The Platform must have the Private Key pair to its announced Public Key and a set of Subject identifiers (subject_id) for the set of users that will be given access to the Tool.

› 5.1.1 OpenID Connect Launch Flow Overview

The Implicit Flow exchange consists of the 6 steps as defined in Section 3.2.1 of OpenID Connect Core [OPENID-CCORE]. However, in order to stop the workflow being vulnerable to the 'Login Cross-Site Request Forgery (CSRF)' each message request MUST be treated as a 3rd party initiated login, as defined in OpenID Connect: Initiating Login from a Third Party, which triggers an OpenID Connect Implicit flow between the tool, acting as the Relying Party, and the platform, acting as the Identity Provider.

In 3rd party initiated login, the login flow is initiated by an OpenID Provider or another party, rather than the Relying Party. In this case, the initiator redirects to the Relying Party at its login initiation endpoint, which requests that the Relying Party send an Authentication Request to a specified OpenID Provider. This login initiation endpoint can be a deep link at the Relying Party, rather than a default landing page.

The OpenID Connect launch flow is shown in Figure 7.

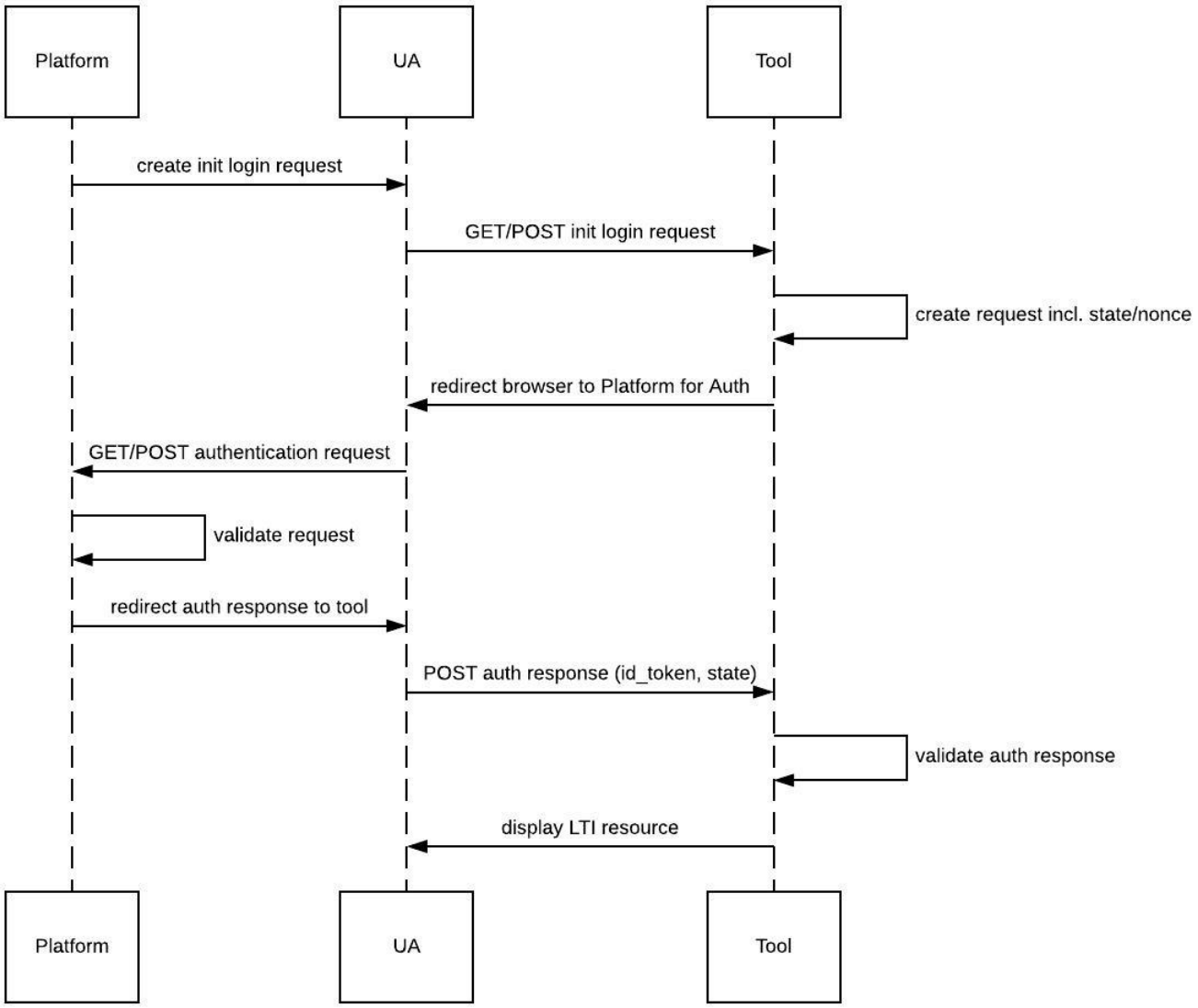

Figure 7 The OpenID Connect launch flow.

As part of the out-of-band registration process the following information must also be distributed:

- The Tool must provide the single URL that the Platform will use to initiate the OpenID Connect authorization flow (called the '3rd Party initiated login end-point');
- The Tool must provide one or multiple redirect URIs that are valid end points where the authorization response can be sent. The number of distinct redirect URIs to be supported by

a platform is not specified;

- The Platform must provide the single end-point to which the browser will be redirected, by the Tool, to initiate the OpenID Connect authentication flow (called the OIDC Authorization end-point').

› *5.1.1.1 Step 1: Third-party Initiated Login*

When a user wants to launch into a `Tool`, the `Platform` will start the OpenID Connect flow by redirecting the User Agent (UA) to the 3rd party initiated login end point. The redirect may be a form POST or a GET - a Tool must support either case, with the following parameters.

iss
REQUIRED. The issuer identifier identifying the learning platform.

login_hint
REQUIRED. Hint to the Authorization Server about the login identifier the End-User might use to log in. The permitted values will be defined in the host specification.

target_link_uri
REQUIRED. The actual end-point that should be executed at the end of the OpenID Connect authentication flow.

NOTE: Other parameters MAY be supplied to provide important context for a specific message exchange i.e. as defined in the corresponding IMS specification.

› *5.1.1.2 Step 2: Authentication Request*

The tool must then perform an authentication request as defined in Section 3.1.2.1 of the [OPENID-CCORE]. The tool sets the CSRF token and binds it to a `state` parameter, and redirects the UA to the platform at the OIDC Authentication endpoint registered with the `iss` with the following parameters:

scope: openid
REQUIRED as per [OPENID-CCORE].

response_type: id_token
REQUIRED as per [OPENID-CCORE].

client_id
REQUIRED as per [OPENID-CCORE]. The Tool's Client ID for this issuer.

redirect_uri
REQUIRED as per [OPENID-CCORE]. One of the registered redirect URIs.

login_hint

REQUIRED. As passed in the initiate login request.

state
RCOMMENDED as per [OPENID-CCORE]. Opaque value for the platform to maintain state between the request and callback and provide Cross-Site Request Forgery (CSRF) mitigation.

response_mode: form_post
REQUIRED. The Token can be lengthy and thus should be passed over as a form POST.

nonce
REQUIRED. String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token.

prompt: none
REQUIRED. Since the message launch is meant to be sent from a platform where the user is already logged in. If the user has no session, a platform must just fail the flow rather than ask the user to log in.

› *5.1.1.3 Step 3: Authentication Response*

Once the platform has validated the `redirect_uri` as a valid end point for the `client_id`, and the current logged in user matches the `login_hint`, the platform can construct the `id_token`.

The `id_token` needs to contain, in addition to the user's identity claims, the message claims required for the tool to fulfill the launch request.

The authentication response is sent to the `redirect_uri` with the following parameters:

state
REQUIRED - see Section 3.2.2.5 of the [OPENID-CCORE].

id_token
REQUIRED - see Section 3.2.2.5 of the [OPENID-CCORE]. This also contains the other message specific claims and the nonce passed in the auth request.

› *5.1.1.4 Step 4: Resource is displayed*

The tool, after decoding and validating the `id_token` and verified the `state` matches the current state attached to the browser session, will display the resource, by, for example, issuing a redirect to the `target_link_uri` as received in the first step of that flow, concluding the Message request.

› *5.1.1.5 Authentication Error Response*

As per Section 3.1.2.6 of the OpenID Connect specification [OPENID-CCORE].

› 5.1.2 ID Token

In order to enable the authentication of end users, OpenID Connect extends OAuth 2.0 with the ID Token data structure. The ID Token is a security token that contains Claims about the Authentication of an end user made by an authorization server when using a Tool, and potentially other requested Claims. The ID Token is represented as a JWT [RFC7519] and the JWS is produced using the Platform's Private Key. See Section 2 of [OPENID-CCORE] for more information regarding the ID Token.

The ID Token uses these Claims:

iss
: REQUIRED. Issuer Identifier for the Issuer of the message i.e. the Platform. The `iss` value is a case-sensitive URL using the HTTPS scheme that contains: scheme, host; and, optionally, port number, and path components; and, no query or fragment components.

aud
: REQUIRED. Audience(s) for whom this ID Token is intended i.e. the Tool. It MUST contain the OAuth 2.0 client_id of the Tool as an audience value. It MAY also contain identifiers for other audiences. In the general case, the `aud` value is an array of case-sensitive strings. In the common special case when there is one audience, the `aud` value MAY be a single case-sensitive string.

sub
: REQUIRED. Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the end user, which is intended to be consumed by the Tool, (for example, it might be something like 24400320 or AItOawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4). It MUST NOT exceed 255 ASCII characters in length. The `sub` value is a case-sensitive string. This MUST be the same value as the Platform's User ID for the end user.

exp
: REQUIRED. Expiration time on or after which the Tool MUST NOT accept the ID Token for processing. When processing this parameter, the Tool MUST verify that the time expressed in this Claim occurs after the current date/time. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. This Claim's value MUST be a JSON number representing the number of seconds offset from 1970-01-01T00:00:00Z (UTC). See [RFC3339] for details regarding date/times in general and UTC in particular.

iat
: REQUIRED. Time at which the Issuer generated the JWT. Its value is a JSON number

representing the number of seconds offset from 1970-01-01T00:00:00Z (UTC) until the generation time.

nonce
REQUIRED. String value used to associate a Tool session with an ID Token, and to mitigate replay attacks. The nonce value is a case-sensitive string.

azp
OPTIONAL. Authorized party - the party to which the ID Token was issued. If present, it MUST contain the OAuth 2.0 Tool ID of this party. This Claim is only needed when the Token has a single audience value and that audience is different than the authorized party. It MAY be included even when the authorized party is the same as the sole audience. The `azp` value is a case-sensitive string containing a String or URI value.

The Claims in an ID Token JWT might look like this (non-normative) example:

```
{
    "iss": "https://lms.uofexample.edu",             // Platform Issuer
Identifier
    "sub": "24400320",                               // Subject Identifier of
the User
    "aud": "s6BhdRkqt3",                             // Client Identifier for
the Client
    "nonce": "n-0S6_WzA2Mj",
    "exp": 1311281970,
    "iat": 1311280970
}
```

The User information passed upon launch depends upon relevant privacy settings the Platform must apply. At a minimum, the Platform must present a permanent identifier for the user in the `sub` Claim.

› 5.1.3 Authentication Response Validation

Tools MUST validate the ID Token in the token response in the following manner:

1. The Tool MUST Validate the signature of the ID Token according to JSON Web Signature [RFC7515], Section 5.2 using the Public Key from the Platform;
2. The Issuer Identifier for the Platform MUST exactly match the value of the `iss` (Issuer) Claim (therefore the Tool MUST previously have been made aware of this identifier);
3. The Tool MUST validate that the `aud` (audience) Claim contains its client_id value registered as an audience with the Issuer identified by the `iss` (Issuer) Claim. The `aud` (audience) Claim MAY contain an array with more than one element. The Tool MUST reject the ID Token if it

does not list the client_id as a valid audience, or if it contains additional audiences not trusted by the Tool. The request message will be rejected with a HTTP code of 401;

4. If the ID Token contains multiple audiences, the Tool SHOULD verify that an `azp` Claim is present;
5. If an `azp` (authorized party) Claim is present, the Tool SHOULD verify that its client_id is the Claim's value;
6. The `alg` value SHOULD be the default of RS256 or the algorithm sent by the Tool in the id_token_signed_response_alg parameter during its registration. Use of algorithms other that RS256 will limit interoperability;
7. The current time MUST be before the time represented by the `exp` Claim;
8. The Tool MAY use the `iat` Claim to reject tokens that were issued too far away from the current time, limiting the amount of time that it needs to store nonces used to prevent attacks. The Tool MAY define its own acceptable time range;
9. The ID Token MUST contain a `nonce` Claim. The Tool SHOULD verify that it has not yet received this nonce value (within a Tool-defined time window), in order to help prevent replay attacks. The Tool MAY define its own precise method for detecting replay attacks.

› 5.2 Tool-Originating Messages

When a Message does not assert the user identity, the sender signs and secures the JWT using the JSON Web Signature (JWS) defined in [RFC7515]. The Tool typically sends such messages to the Platform. Figure 8 is a schematic representation of this architecture.

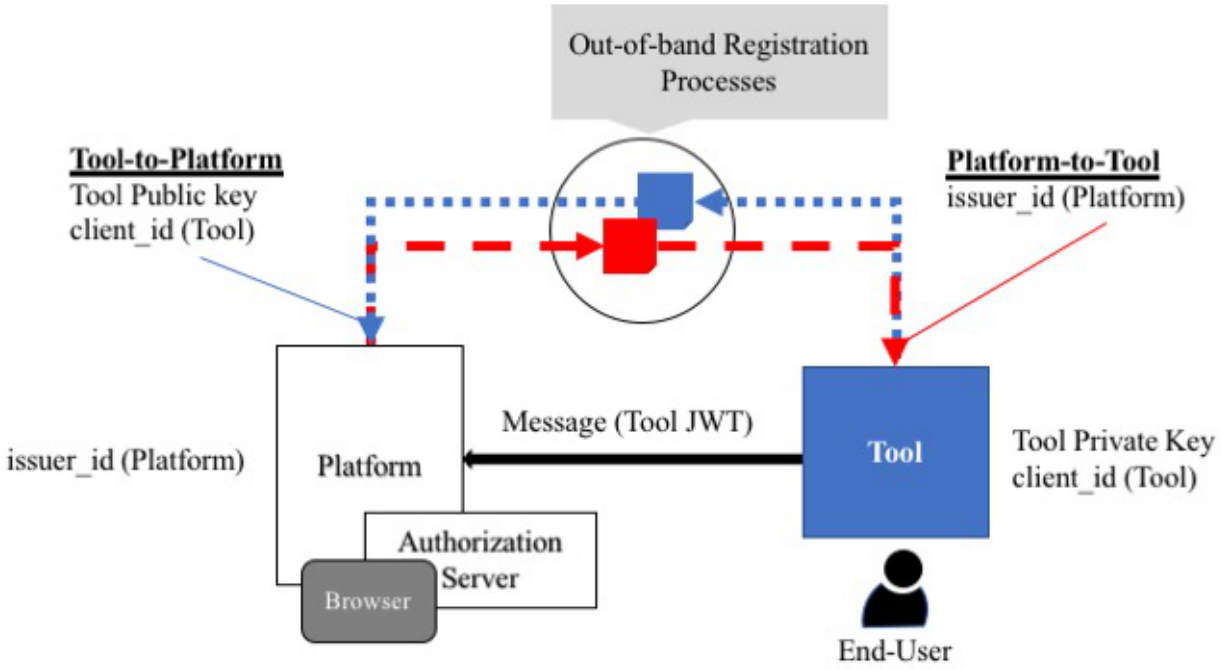

Figure 8 Information flow for tool-originating messages.

The key points in Figure 8 are:

- The Tool must have been allocated an OAuth 2 client_id (usually by an Authorization Server and this may be owned by the Platform)
- The Platform must be given the Public Key and client_id for the Tool (using some out-of-band registration process if not assigned by the Platform). See 6. Key Management for more details on Key Management.
- The Tool must be given the Issuer Identifier (issuer_id) for the Platform (using some out-of-band registration process)
- The Tool must have the Private Key pair to its announced Public Key.

› 5.2.1 Form Parameter

The sender sends the JWT via a form HTTP POST with a single parameter:

JWT (REQUIRED). JWS-signed token containing the Message Claims. The JWS is produced using the Tool's Private Key.

› 5.2.2 Tool JWT

The Tool JWT is a security token which contains Claims about the Authentication of an end user

made by an authorization server when using a Tool, and potentially other requested Claims. The Tool JWT is represented as a JWT [RFC7519].

The Tool JWT uses these claims:

iss
: REQUIRED. Issuer Identifier for the Issuer of the message i.e. the Tool. It must be the OAuth 2.0 client_id of the Tool (this MAY be provided to it by the Platform upon registration of the Tool).

aud
: REQUIRED. Audience(s) for whom this Tool JWT is intended. It MUST contain the case-sensitive URL used by the Platform to identify itself as an Issuer in platform-originating Messages. In the common special case when there is one audience, the `aud` value MAY be a single case-sensitive string.

exp
: REQUIRED. Expiration time on or after which the Platform MUST NOT accept the Tool JWT for processing. When processing this parameter, the Platform MUST verify that the time expressed in this Claim occurs after the current date/time. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. This Claim's value MUST be a JSON number representing the number of seconds offset from 1970-01-01T00:00:00Z (UTC). See [RFC3339] for details regarding date/times in general and UTC in particular.

iat
: REQUIRED. Time at which the Issuer generated the Tool JWT. Its value is a JSON number representing the number of seconds offset from 1970-01-01T00:00:00Z (UTC) until the generation time.

nonce
: REQUIRED. String value used to associate a Tool session with a Tool JWT, and to mitigate replay attacks. The nonce value is a case-sensitive string.

azp
: OPTIONAL. Authorized party - the party to which the Tool JWT was issued. If present, it MUST contain the same value as in the `aud` Claim. The `azp` value is a case-sensitive string containing a String or URI value.

The Claims in a Tool JWT might look like this (non-normative) example:

```
{
    "iss": "s6BhdRkqt3",                                  // Client Identifier of
the Client
    "aud": "https://lms.uofexample.edu",                  // Platform Issuer
Identifier
    "nonce": "n-0S6_WzA2Mj",
    "exp": 1311281970,
```

```
    "iat": 1311280970
}
```

› 5.2.3 Authentication Response Validation

Platforms MUST Validate the Message Tool JWT in the Token Response in the following manner:

1. The Platform MUST validate the signature of the Tool JWT according to JSON Web Signature [RFC7518] using the algorithm the Tool specifies in the `alg` header parameter of the JOSE Header. The Platform must use the Public Key from the Tool to validate the message;
2. The client_id for the Tool MUST exactly match the value of the `iss` (Issuer) Claim;
3. The Platform MUST validate that the `aud` (audience) Claim contains its advertised Issuer URL. The Platform must reject the Tool JWT if it does not list the Platform as a valid audience, or if it contains additional audiences not trusted by the Platform;
4. If the Token contains multiple audiences, the Platform SHOULD verify that an `azp` Claim is present;
5. If an `azp` (authorized party) Claim is present, the Platform SHOULD verify that its Issuer URL is the Claim Value.
6. The `alg` value SHOULD be the default of RS256 or the algorithm specified by the Platform to the Tool during registration;
7. The current time MUST be before the time represented by the `exp` Claim;
8. The Platform MAY use the `iat` to reject tokens that were issued to far away from the current time, limiting the amount of time that it needs to store nonces used to prevent attacks. The Platform MAY define its own acceptable time range;
9. The Tool JWT MUST contain a `nonce` Claim. The Platform SHOULD verify that it has not yet received this nonce value (within a Platform-defined time window), in order to prevent replay attacks. The Platform MAY define its own precise method for detecting replay attacks.

› 5.3 Message Specific Claims

Message Tool JWTs MAY contain other Claims. The receiver of a Message Tool JWT MUST ignore any claims it does not understand. Vendors MAY extend the Message Tool JWT by adding additional Claims using a "Public Claim Name" as defined in Section 4.2 of the JWT in the [RFC7519] specification. Vendors SHOULD only use domains that they own, and they MUST prefix these domains with "http://".

› 5.4 Message Signing

Message Tool JWTs MUST be signed using the method described in [RFC7518], Message Tool JWTs MUST NOT use `none` as the `alg` value.

Message Client JWTs SHOULD NOT use the JWS x5u, x5c, jku, or jwk Header Parameter fields. Instead, Platforms and Tools should communicate the keys to use for Message JWS Tokens during registration.

› 6. Key Management

Some systems will require key management. In cases where systems use asymmetric keys, the Issuer of a JWT or access token signs it with its private key, and the recipient verifies the signature by using the Issuer's public key. The Issuer could be either a Platform or a Consumer. The system responsible for originating the message is the Issuer. The mechanisms by which keys are minted and distributed is outside the scope of this framework (see the Best Practice Recommendations). Therefore, there must be an out-of-band registration process during which access to the public keys is supplied. It should be noted that a Platform could be responsible for allocating key-sets to a Client: the integrity of the private key MUST be maintained by the accompanying distribution mechanism.

› 6.1 RSA Key

Where systems use RSA Keys, they MUST use SHA-256 (RS256) as a minimum as defined in [RFC7518]. Support for other algorithms is permitted but their use limits interoperability. Later versions of this framework MAY add OPTIONAL support for other algorithms.

› 6.2 JSON Web Key

When systems use JSON Web Keys (JWK) to represent the public key, such representations SHOULD conform to [RFC7517]: parties exchanging keys represented this way MUST use this form during key exchanges.

When using RSA keys, they MUST include the ***n*** (modulus) and ***e*** (exponent) as defined in [RFC7518] (see the key set example in Section 6.3 Key Set URL).

› 6.3 Key Set URL

When systems use Key Sets, they MUST provide a URL to the key set (the system responsible for

supplying this URL must be identified in the corresponding IMS service specification). A JWK Set is a container of one or multiple public keys identified by their key identifier (`kid` key parameter). The [RFC7517] defines JWK Sets.

The supplier of the key set URL MUST use the `kid` parameter to identify the keys. Even when there is only one key in a key-set a `kid` MUST be supplied. Both a Platform and a Consumer could use key-sets.

The Issuer of a JWT identifies the key a receiver uses to validate the JWT signature by using the `kid` JWT header Claim. The Issuer MUST NOT reuse the `kid` identifier to identify different public keys of the same type (`kty`), allowing the public key to be cached by the Consumer. The Issuer MAY issue a `cache-control: max-age` HTTP header on requests to retrieve a key set to signal how long the retriever may cache the key set before refreshing it.

Here is an example of a JWK Set containing two RSA keys:


```
{
    "keys": [{
        "e": "AQAB",
        "use": "sig",
        "alg": "RS256",
        "kty": "RSA",
        "n": "oNqXxxWuX7LlovO5reRNau5f96K_o3DJx-wK7lrjBmp0qKwNszbbp8MvfrlVs-
oYXfj1rzqAeY6GJF5BETViDTT0i2fEz37J0HGAeTrO7Z5zI5Ure9Cb0lulLOZj1hF8piZzWW_z_set2N
yhafoZ-IG1NSe61mqHu7mTjuHYST84igz-
bPKhkJAVlmPPjHTO51hG9T_roVIkjXnvgqd2dCaJ0ExT2bR96jcyausbkdDNfPtJdfSCAWYXGQnt0PmI
ysOHPtCkyFqv5ez8KXT7Q4CAYd7nxwfWNOFRHyLAyF__cYEJlBEKGyJniSIPtkGBWrbXUQhKF6TEFa-
RRRl8Dw==",
        "kid": "1516918956_0"
    }, {
        "e": "AQAB",
        "use": "sig",
        "alg": "RS256",
        "kty": "RSA",
        "n": "kMfHwTp2dIYybtvU-xzF2E3dRJBNBtNbb-d3-
Rm6nRUraxnTwZ6Fr1YpFBd1pnWzLzdtMv7ofCd28nx-1mfYZ6qhqPWF1RpGe2vVOSTmcu-QpA9h-
rouqRKlv3jvXPn623Z2U1Wml0EIxyIzD3WLu7NkWEKSIcBzeY1TctpO5FSU3EyyCX1UoIMuvYBP9tiZl
c74yIZvky-
qT8Ej3S8L0JqhvD583E_uGMoLowguOl2yYr9zhubiqOxT3VsxvpJCu04TWmvf4XX34IQRyAHcPJFQ2Qi
BfLWvWyc6iP3JJYJvyapwc5vVEismryXnngyBX8NXHZaarMi6g5kTQi8itw==",
        "kid": "1516918956_1"
    }]
}
```


The details of the permitted parameters for RSA keys is supplied in Section 6.3 of [RFC7518]. This RFC should also be used for permitted parameters when using other JSON Web Algorithms.

› 6.4 Issuer Public Key Rotation

When the Issuer rotates its public key, the Issuer MUST add it to the JSON Key Set under a new `kid`. Other parties can then download a new version of the JSON Key Set. IMS recommends that Issuers doing key rotation preserve the previous public key in the JSON Key Set to allow an overlap.

If the Issuer does not use the `kid` parameter to identify its key, other parties using the key SHOULD use the cache-control header to properly rotate their cached copies of the key.

› 7. Best Practice Recommendations

Implementors should ensure that they are familiar with the OAuth 2.0 threat model and security considerations addressed in [RFC6819]. It is also RECOMMENDED that implementors are aware of the latest OAuth 2.0 security best current practices [OAUTH2-SBP] and similarly for the use of JSON Web Tokens [JSONWT-BP].

› 7.1 Access Token Management

› 7.1.1 Expires_In Values and Renewing the Access Token

The recommended value of the 'expires_in' attribute is 3600 i.e. one hour. This means that the validity of the access token expires one hour after the time it was issued. Client-credentials based OAuth 2 does NOT permit the use of access token refreshing. Therefore, once an access token has expired, a new access token MUST be requested. The same set of credentials MAY be reused when requesting a new access token. However, these credentials MAY also expire: if they expire then a request for an access token using these credentials MUST be refused. The use of expiry times for credentials is implementation dependent.

› 7.1.2 Authorization Code Details

The Authorization Code MUST be used only once. A lifetime for the authorization code of 600 seconds (10 minutes) is RECOMMENDED. If an authorization code is used more than once, the authorization server MUST deny the request and SHOULD revoke (when possible) all tokens previously issued based on that authorization code. The authorization code is bound to the client identifier and redirection URI

› 7.1.3 Scope Naming Conventions

When requesting an access token it is a requirement, for access to an IMS-compliant service, to identify the set of scopes for the access request. The set of scopes that are available as part of an IMS service are defined in the corresponding specification document. In this document is the naming convention that SHOULD be adopted for those scopes. This naming convention is based upon a broader set of guidelines created by IMS for all of its generated artifacts. The format for a scope is:

```
https://purl.imsglobal.org/spec/[shortname]/[version]/scope/[scopeleaf].[action]
```

where:

[shortname] The abbreviated name of the service specification e.g. 'or' for OneRoster, 'lti' for LTI, etc.

[version] The version of the service specification e.g. 'v1p0, 'v2p1', etc.

[scopeleaf] An appropriate term for the collection of endpoints being covered by the scope.

[action] A term that reflects that nature of the scope. The suggested values are:

- 'readonly' for a set of endpoints that permit read only using the 'GET' verb
- 'createput' for a set of endpoints that permit creation using the 'PUT' verb
- 'createpost' for a set of endpoints that permit creation using the 'POST' verb
- 'update' for a set of endpoints that permit changing of an established resource
- 'replace' for a set of endpoints that permit overwriting of an established resource
- 'delete' for a set of endpoints that permit delete only
- 'all' for access to all of endpoints supported for the version of the identified specification

An example of some scopes that have already been defined by IMS are:

```
https://purl.imsglobal.org/spec/or/v1p2/scope/gradebook.delete
```

which is used in OneRoster 1.2 to permit access to the endpoints that allow a gradebook resource to be deleted, and:

```
https://purl.imsglobal.org/spec/rs/v1p0/scope/resource.readonly
```

which is used in LTI Resource Search 1.0 to permit access to the endpoints that allow access to the information about a set of resources.

› 7.1.4 Managing Scopes

IMS REQUIRES the use of scopes when obtaining an access token. The definition of the set of scopes for a service is contained within the corresponding IMS specification. Therefore, an authorization server MUST be made aware of these scopes. Either the credentials or the JWT claim set supplied when requesting the access token must be used to determine if the requesting client is permitted to request access depending on those scopes. Scope definitions are immutable and permanent.

The way in which an authorization server obtains the information about the set of permitted scopes for a service is implementation dependent. An authorization server MUST support validation of scopes with respect to access token provision. A scope MUST NOT be allocated if it is not contained within the request for the access token.

› 7.2 Key Distribution

The IMS approach to message signing is based upon the use of key-pairs (private and public key) by the two end-systems i.e. the Platform and the Consumer. Whereas distribution of public keys does not require a secure mechanism, the integrity of the private keys must not be compromised. A system SHOULD NOT use a single key pair to secure message signing for more than one system. Therefore, systems SHOULD be capable of obtaining and using many key pairs and MAY use key exchange and rotation using a JWK Set and Key Set URLs to manage the use of many key pairs.

Let the private and public key pair for a Platform be denoted by { P[X], P[K] } and the equivalent for a Consumer by { C[X], C[K] }. In the case of key-sets this becomes {{ P[X1], P[K1] }, ..., { P[Xm], P[Km] }} and {{ C[X1], C[K1] }, ..., { C[Xn], C[Kn] }} i.e. the are 'm' key-pairs for the Platform and 'n' key-pairs for the Consumer. Knowledge of the public keys, P[X1]..P[Km] and C[X1]..C[Kn] is unrestricted. Knowledge of the private keys SHOULD be as restricted as possible. For a Platform/Consumer pair the keys can be created by:

Platform only. The Platform creates both sets of key-pairs. The Platform is responsible for providing the Consumer with its keys i.e. the Platform creates all of the keys and MUST securely give the Consumer the keys {{ C[X1], C[K1] }, ..., { C[Xn], C[Kn] }}. The Consumer must also be given the Platform's public keys i.e. { P[K1]..P[Km] }. The advantage of this approach is that the Consumer is not burdened with key creation and distribution.

Consumer only. The Consumer creates both sets of key-pairs. The Consumer is responsible for providing the Platform with the private key i.e. the Consumer creates all of the keys and MUST securely give the Platform the keys {{ P[X1], P[K1] }, ..., { P[Xn], P[Km] }}. The Platform must also be given the Client's public keys i.e. { C[K1]..C[Kn] }. The disadvantage of this approach is that the Platform is dependent upon the Consumer creating sufficiently robust keys and for ensuring the

integrity of these keys is not compromised.

Platform and Consumer Independently. Each system creates their on key pairs. Each system is responsible for making the other aware of the public keys. The advantage of this approach is that only the user of the private keys has knowledge of those keys. The disadvantage is that the Consumer functionality is more complex.

In general, there will be many more Consumers than Platforms. It is important to minimise the implementation effort required to achieve the required message signing. This security framework is based upon security standards that have broad adoption. Key generation and distribution between Platforms and Consumers has not been defined. The ways in which key distribution is managed within a Consumer is dependent on:

- The business model used for the provision of access to the Consumer. The distribution modes for the Consumer itself will indicate the best ways to manage the corresponding allocation of private keys and access to the relevant public keys;
- The key distribution models that become preferred by the Platforms. For educational technology this will be determined by the approaches required by learning management systems and/or student information systems.

New requirements and recommendations MAY be made in a later version of this framework, if, at some later date, there is a clarification, in the education technology marketplace, on the preferred mechanism for key distribution.

› 7.3 Handling Security Vulnerabilities

› 7.3.1 Prohibiting the Login CSRF Vulnerability

For Platform initiated message exchanges the combination of the full six steps of the 'Implicit Flow' with the 'Initiating Login from a Third Party' mechanisms of the [OPENID-CCORE] prevents vulnerability to Login CSRF attacks. This vulnerability does NOT occur in Tool initiated exchanges because the user is already logged into the platform. The Platform must verify that the data claim returned in the Tool initiated message was generated by the current user's session in a corresponding Tool launch.

› 7.3.2 Symmetric vs. Asymmetric Keys with JWT

The primary difference between the use of Symmetric and Asymmetric keys is that the same key is used in both end-systems when using Symmetric keys. When using Asymmetric keys the end-systems have their own public and private key pairs. IMS RECOMMENDS the use of Asymmetric keys. If symmetric keys are used then the associated IMS service specification MUST explain and

justify why that approach has been adopted. Asymmetric key usage should reduce a system's vulnerability but the overall vulnerability of a system is based upon the combination of many issues.

A. Relevant Standards Summaries

A.1 Relevant Request for Comments

A.1.1 RFC 2616 - HyperText Transfer Protocol

The Hypertext Transfer Protocol (HTTP/1.1) [RFC2616] is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol that can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification defines the protocol referred to as “HTTP/1.1”, and is an update to [RFC2068].

This document is available at: https://tools.ietf.org/pdf/rfc2616.pdf.

A.1.2 RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication

"HTTP/1.0", includes the specification for a Basic Access Authentication scheme [RFC2617]. This scheme is not considered to be a secure method of user authentication (unless used in conjunction with some external secure system such as SSL), as the user name and password are passed over the network as clear text.

This document also provides the specification for HTTP’s authentication framework: the original Basic authentication scheme, and a scheme based on cryptographic hashes referred to as "Digest Access Authentication". It is, therefore, also intended to serve as a replacement for [RFC2069]. Some optional elements specified by RFC 2069 have been removed from this specification due to problems found since its publication; other new elements have been added for compatibility--those new elements have been made optional, but are strongly recommended.

Like Basic authentication, Digest Access authentication verifies that both parties to a communication know a shared secret (a password); unlike Basic authentication, this verification can be done without sending the password in the clear, which is Basic authentication’s biggest weakness. As with most other authentication protocols, the greatest sources of risks are usually found not in the core protocol itself but in policies and procedures surrounding its use.

This document is available at: https://tools.ietf.org/pdf/rfc2617.pdf.

› A.1.3 RFC 4949 - Internet Security Glossary Version 2

This Glossary provides definitions, abbreviations, and explanations of terminology for information system security [RFC4949]. The 334 pages of entries offer recommendations to improve the comprehensibility of written material that is generated in the Internet Standards Process [RFC2026]. The recommendations follow the principles that such writing should follow these guidelines:

a. Use the same term or definition whenever the same concept is mentioned.

b. Use terms in their plainest, dictionary sense.

c. Use terms that are already well-established in open publications.

d. Avoid terms that either favor a particular vendor or favor a particular technology or mechanism over other, competing techniques that already exist or could be developed.

This document is both a major revision and a major expansion of the original Internet Security Glossary [RFC2828]. This revised Glossary is an extensive reference that should help the Internet community to improve the clarity of documentation and discussion in an important area of Internet technology. However, readers should be aware of the following points:

a. The recommendations and some particular interpretations in definitions are those of the author, not an official IETF position. The IETF has not taken a formal position either for or against recommendations made by this Glossary, and the use of [RFC2119] language (e.g. SHOULD NOT) in the Glossary must be understood as unofficial. In other words, the usage rules, wording interpretations, and other recommendations that the Glossary offers are personal opinions of the Glossary's author. Readers must judge for themselves whether or not to follow his recommendations, based on their own knowledge combined with the reasoning presented in the Glossary.

b. The Glossary is rich in the history of early network security work, but it may be somewhat incomplete in describing recent security work, which has been developing rapidly.

This document is available at: https://tools.ietf.org/pdf/rfc4949.pdf.

› A.1.4 RFC 5246 - The Transport Layer Security Protocol Version 1.2

This document specifies Version 1.2 of the Transport Layer Security (TLS) protocol [RFC5246]. The TLS protocol provides communications security over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

This document is available at: https://tools.ietf.org/pdf/rfc5246.pdf.

› A.1.5 RFC 5849 - The OAuth 1.0 Protocol

OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different Client or an end user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.

Use of OAuth 1.0 [RFC5849] within the context of IMS specifications is deprecated. The material in this document explains the security mechanisms that MUST be used to replace OAuth 1.0.

This document is available at: https://tools.ietf.org/pdf/rfc5849.pdf.

› A.1.6 RFC 6749 - The OAuth 2.0 Authorization Framework

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification [RFC6749] replaces and obsoletes the OAuth 1.0 protocol described in [RFC5849].

This document is available at: https://tools.ietf.org/pdf/rfc6749.pdf.

› A.1.7 RFC 6750 - The OAuth 2.0 Authorization Framework Bearer Token Usage

This specification describes how to use bearer tokens in HTTP requests to access OAuth 2.0 protected resources. Any party in possession of a bearer token (a "bearer") can use it to get access to the associated resources (without demonstrating possession of a cryptographic key). To prevent misuse, bearer tokens need to be protected from disclosure in storage and in transport [RFC6750].

This document is available at: https://tools.ietf.org/pdf/rfc6750.pdf.

› A.1.8 RFC 6819 - OAuth 2.0 Threat Model and Security Considerations

This document gives additional security considerations for OAuth, beyond those in the OAuth 2.0 specification, based on a comprehensive threat model for the OAuth 2.0 protocol.

This document is available at: https://tools.ietf.org/pdf/rfc6819.pdf.

› A.1.9 RFC 7515 - JSON Web Signature (JWS)

JSON Web Signature (JWS) represents content secured with digital signatures or Message Authentication Codes (MACs) using JSON-based data structures [RFC7515]. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and an IANA registry defined by that specification. Related encryption capabilities are described in the separate JSON Web Encryption (JWE) specification.

This document is available at: https://tools.ietf.org/pdf/rfc7515.pdf.

› A.1.10 RFC 7517 - JSON Web Key

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. This specification also defines a JWK Set JSON data structure that represents a set of JWKs [RFC7517]. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and IANA registries established by that specification.

This document is available at: https://tools.ietf.org/pdf/rfc7517.pdf.

› A.1.11 RFC 7518 - JSON Web Algorithms

This specification registers cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS), JSON Web Encryption (JWE), and JSON Web Key (JWK) specifications [RFC7518]. It defines several IANA registries for these identifiers.

This document is available at: https://tools.ietf.org/pdf/rfc7518.pdf.

› A.1.12 RFC 7519 - JSON Web Token

A JWT is a compact, URL-safe means of representing Claims to be transferred between two parties. The Claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the Claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted [RFC7519].

This document is available at: https://tools.ietf.org/pdf/rfc7519.pdf.

› A.1.13 RFC 7523 - JSON Web Token Profile for OAuth 2.0 Client Authentication and Authorization Grants

This specification defines the use of a JWT bearer token as a means for requesting an OAuth 2.0

access token as well as for client authentication [RFC7523].

This document is available at: https://tools.ietf.org/pdf/rfc7523.pdf.

› A.1.14 RFC 7636 - Proof Key for Code Exchange by OAuth Public Clients

OAuth 2.0 public clients utilizing the Authorization Code Grant are susceptible to the authorization code interception attack. This specification describes the attack as well as a technique to mitigate against the threat through the use of Proof Key for Code Exchange (PKCE, pronounced "pixy").

This document is available at: https://tools.ietf.org/pdf/rfc7636.pdf.

› A.1.15 RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3

This document specifies version 1.3 of the Transport Layer Security (TLS) protocol. TLS allows client/server applications to communicate over the Internet in a way that is designed to prevent eavesdropping, tampering, and message forgery.

This document updates RFCs 5705 and 6066, and obsoletes RFCs 5077, 5246, and 6961. This document also specifies new requirements for TLS 1.2 implementations.

This document is available at: https://tools.ietf.org/html/rfc8446.

› A.2 Relevant Other Standards

› A.2.1 OpenID Connect Core

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables Clients to verify the identity of the end user based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the end user in an interoperable and REST-like manner. This specification defines the core OpenID Connect functionality: authentication built on top of OAuth 2.0 and the use of Claims to communicate information about the end user. It also describes the security and privacy considerations for using OpenID Connect [OPENID-CCORE].

This document is available at: http://openid.net/specs/openid-connect-core-1_0.html.

› A.2.2 OAuth 2.0 Form Post Response Mode

This specification defines the Form Post Response Mode. In this mode, Authorization Response

parameters are encoded as HTML form values that are auto-submitted in the User Agent, and thus are transmitted via the HTTP POST method to the Client, with the result parameters being encoded in the body using the application/x-www-form-urlencoded format [OAUTH2-FPRM].

This document is available at: http://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html.

› B. Revision History

This section is non-normative.

› B.1 Version History

IMS Final Release 1.0	15 May 2019	The first formal Final Release of this document.

› C. References

› C.1 Normative references

[ISO29115]
Information technology - Security techniques - Entity authentication assurance framework. URL: https://www.iso.org/standard/45138.html

[ITU-X1252]
X.1252: Baseline identity management terms and definitions. 2010. URL: https://www.itu.int/rec/T-REC-X.1252-201004-I

[OAUTH2-FPRM]
OAuth 2.0 Form Post Response Mode. IETF. April 27, 2015. URL: http://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html

[OPENID-CCORE]
OpenID Connect Core 1.0. Nov 8 2014. URL: http://openid.net/specs/openid-connect-core-1_0.html

[RFC2069]
An Extension to HTTP : Digest Access Authentication. J. Franks; P. Hallam-Baker; J. Hostetler; P. Leach; A. Luotonen; E. Sink; L. Stewart. IETF. January 1997. Proposed Standard. URL: https://tools.ietf.org/html/rfc2069

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC2616]

Hypertext Transfer Protocol -- HTTP/1.1. R. Fielding; J. Gettys; J. Mogul; H. Frystyk; L. Masinter; P. Leach; T. Berners-Lee. IETF. June 1999. Draft Standard. URL: https://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. IETF. June 1999. Draft Standard. URL: https://tools.ietf.org/html/rfc2617

[RFC3339]

Date and Time on the Internet: Timestamps. G. Klyne; C. Newman. IETF. July 2002. Proposed Standard. URL: https://tools.ietf.org/html/rfc3339

[RFC4949]

Internet Security Glossary, Version 2. R. Shirey. IETF. August 2007. Informational. URL: https://tools.ietf.org/html/rfc4949

[RFC5246]

The Transport Layer Security (TLS) Protocol Version 1.2. T. Dierks; E. Rescorla. IETF. August 2008. Proposed Standard. URL: https://tools.ietf.org/html/rfc5246

[RFC5849]

The OAuth 1.0 Protocol. E. Hammer-Lahav, Ed.. IETF. April 2010. Informational. URL: https://tools.ietf.org/html/rfc5849

[RFC6749]

The OAuth 2.0 Authorization Framework. D. Hardt, Ed.. IETF. October 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6749

[RFC6750]

The OAuth 2.0 Authorization Framework: Bearer Token Usage. M. Jones; D. Hardt. IETF. October 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6750

[RFC6819]

OAuth 2.0 Threat Model and Security Considerations. T. Lodderstedt, Ed.; M. McGloin; P. Hunt. IETF. January 2013. Informational. URL: https://tools.ietf.org/html/rfc6819

[RFC7515]

JSON Web Signature (JWS). M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7515

[RFC7517]

JSON Web Key (JWK). M. Jones. IETF. May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7517

[RFC7518]
JSON Web Algorithms (JWA). M. Jones. IETF. May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7518

[RFC7519]
JSON Web Token (JWT). M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7519

[RFC7523]
JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants. M. Jones; B. Campbell; C. Mortimore. IETF. May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7523

[RFC7636]
Proof Key for Code Exchange by OAuth Public Clients. N. Sakimura, Ed.; J. Bradley; N. Agarwal. IETF. September 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7636

[RFC8446]
The Transport Layer Security (TLS) Protocol Version 1.3. E. Rescorla. IETF. August 2018. Proposed Standard. URL: https://tools.ietf.org/html/rfc8446

› C.2 Informative references

[JSONWT-BP]
JSON Web Token Best Current Practices (draft-ietf-oauth-jwt-bcp-04). IETF. November 08, 2019. Best Current Practice. URL: https://datatracker.ietf.org/doc/draft-ietf-oauth-jwt-bcp/

[LTI-13]
IMS Global Learning Tools Interoperability® Core Specification v1.3. C. Vervoort; N. Mills. IMS Global Learning Consortium. April 2019. IMS Final Release. URL: https://www.imsglobal.org/spec/lti/v1p3/

[OAUTH2-SBP]
OAuth 2.0 Security Best Current Practice (draft-ietf-oauth-security-topics-11). IETF. December 28, 2019. Best Current Practice. URL: https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics/

[RFC2026]
The Internet Standards Process -- Revision 3. S. Bradner. IETF. October 1996. Best Current Practice. URL: https://tools.ietf.org/html/rfc2026

[RFC2068]
Hypertext Transfer Protocol -- HTTP/1.1. R. Fielding; J. Gettys; J. Mogul; H. Frystyk; T. Berners-Lee. IETF. January 1997. Proposed Standard. URL: https://tools.ietf.org/html/rfc2068

[RFC2828]

Internet Security Glossary. R. Shirey. IETF. May 2000. Informational. URL: https://tools.ietf.org/html/rfc2828

› D. List of Contributors

The following individuals contributed to the development of this document:

Andrew Cunningham	Google	
Paul Gray	Learning Objects	
Viktor Haag	D2L	
Dereck Haskins	IMS Global	
Martin Lenord	Turnitin	
Mark Leuba	IMS Global	
Karl Lloyd	Instructure	
Mark McKell	IMS Global	editor
Nathan Mills	Instructure	editor
Padraig O'hiceadha	HMH	
Marc Phillips	Instructure	
Eric Preston	Blackboard	
James Rissler	IMS Global	
Charles Severance	University of Michigan	
Colin Smythe	IMS Global	editor
James Tse	Google	
Claude Vervoort	Cengage	editor

Jim Walkoski D2L