OAuth-based authentication for IPP Print Systems

Document version: 2.00

Design proposed by Google - Chrome OS team

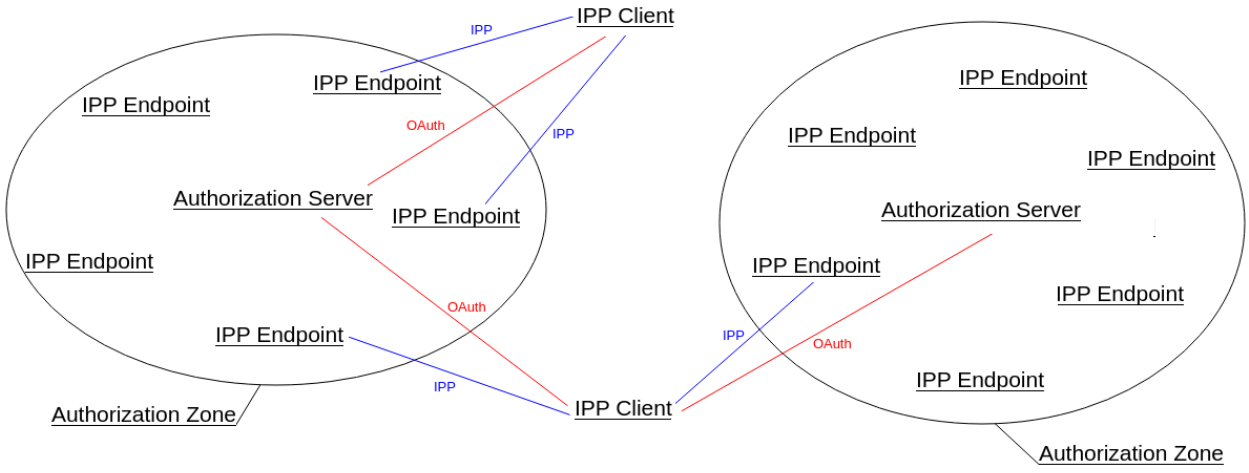
Objective

This document proposes a standard OAuth-based authentication procedure for print systems based on the IPP protocol. The approach proposed here was inspired by the OAuth authentication procedure, described in PWG 5199.10-2019 (see 4.6), and is compatible with the OAuth 2.0 standard. The original procedure proposed in the aforementioned document was modified to address security issues. The goal is to create a well-defined, secure and easy-to-implement authentication interface between print management systems and their clients.

Introduction

Base definitions

The diagram below presents a simplified model of the described system:



IPP Client - a piece of software running on a device controlled by the user. An IPP Client tries to establish communication with a printer (IPP Endpoint) through the IPP protocol to submit a print job. IPP Client may be a part of the operating system, a standalone application, or a library that can communicate directly with a given IPP Endpoint. An IPP client is considered to be a public OAuth Client, while IPP Endpoint is considered a Resource Server (see RFC6749-1.1 for a description of general OAuth roles and RFC6749-2.1 for a definition of public OAuth client).

IPP Endpoint - a network location with which an IPP Client can establish the IPP communication and send a print job to it; it may be a printer or an endpoint provided by a print server or other IPP-based print system. In particular, it may be an Infrastructure Printer as defined in PWG 5100.18-2015.

Authorization Server - a server responsible for coordinating the OAuth authorization process. It controls access to an IPP Endpoint.

Authorization Zone - a set of all IPP Endpoints controlled by the same Authorization Server. ***AuthorizationServerURI*** (an URL of the Authorization Server) can be considered as a unique identifier of the Authorization Zone.

The scope of this document

This document describes an OAuth-based authentication framework for print management systems. The goal is to standardize the way of controlling access to IPP Endpoint by IPP Clients. The proposed solution does not impose any restrictions on the architecture of the Authorization Zone. Print management system may be a single Authorization Zone, or it may consist of many Authorization Zones. The document also does not specify how IPP Clients acquire information about available IPP Endpoints. We assume here that ***IPPEndpointURL***s of available IPP Endpoints may be obtained by an IPP Client in any of the following ways - they may be:

- queried from a print server specified by the user or provided by the administrator
- discovered on the local network with mDNS protocol
- specified directly by the user or provided by the administrator.

The security of the communication between IPP Clients and Authorization Servers or IPP Endpoints is based on TLS. In all cases described in this document, an IPP Client uses https protocol to communicate with the other actors. IPP Clients must always verify the SSL certificate of an Authorization Server and IPP Endpoints before sending any requests to them.

Variables used in the document

IPPEndpointURI - URL of IPP Endpoint.

IPPEndpointSSLFingerprint - a fingerprint of the IPP Endpoint SSL certificate.

AuthorizationServerURI - URL of the Authorization Server.

RegistrationEndpointURI, ***AuthorizationEndpointURI***, ***TokenEndpointURI***, ***RevocationEndpointURI*** - URL provided by the Authorization Server and used as target locations of the corresponding HTTP requests, defined later in this document (HTTP exchanges). They all must use the same SSL certificate as the Authorization Server.

AccessToken - a sequence of bytes issued for an IPP Client after successful completion of the OAuth authorization process. It is a secret of a single OAuth session.

RefreshToken - a sequence of bytes issued for the IPP Client after successful completion of the OAuth authorization process (optional). It may be used for obtaining a new ***AccessToken*** in the same OAuth session.

EndpointAccessTokens - a sequence of bytes issued for an IPP Client to communicate with a particular IPP Endpoint. The IPP Client must have a valid ***AccessToken*** to obtain this token.

ClientID - a unique name assigned to the IPP Client by the Authorization Server.

Scope - see RFC6749-3.3.

Rationale

Existing OAuth authentication protocol for IPP

The OAuth authentication for the IPP system was proposed in section 4.6 of the following document: PWG 5199.10-2019. This section describes the general communication schema between an IPP Client and an IPP Endpoint (a printer). The standard use case can be summarized by the following steps:

1. IPP Client wants to print to a given IPP Endpoint that requires the OAuth authorization
2. IPP Client queries the ***AuthorizationServerURI*** from the IPP Endpoint
3. IPP Client registers itself to the Authorization Server, completes the OAuth authorization process, and obtains ***AccessToken***
4. IPP Client sends a print job to the IPP Endpoint with the obtained ***AccessToken***
5. IPP Endpoint verifies the ***AccessToken*** and prints the document

We will refer to this procedure as “the original protocol”.

Security drawbacks of the original protocol

The aforementioned PWG document does not impose any restrictions on system configuration or the way in which the IPP Client obtains information about available printers (IPP Endpoints). However, these two aspects have a significant impact on the system security. Since there are no restrictions imposed on ***IPPEndpointURI***s, one can try to introduce a fake IPP Endpoint to the system to intercept the ***AccessToken*** or contents of printed documents. A fake IPP Endpoint can also redirect an IPP Client to a fake Authorization Server, which can be used to impersonate the existing Authorization Zone.

To mitigate the risk of disclosing an ***AccessToken*** or a print job to an unauthorized node, an IPP Client must verify that the target IPP Endpoint is a part of the Authorization Zone and use different dedicated access tokens for every IPP Endpoint. To enable this kind of verification we propose here to extend the original protocol by Token Exchange request defined in RFC8693. This request must be used by the IPP Client to obtain dedicated ***EndpointAccessTokens*** for each accessed IPP Endpoint. It allows to implement the following security mechanisms:

- Authorization Server may issue different ***EndpointAccessToken***s for different ***IPPEndpointURI***s. It would prevent reusing the token by other entities.
- During the Token Exchange, the Authorization Server may check if the given ***IPPEndpointURI*** belongs to the Authorization Zone.

The Authorization Server must use one of these approaches or combination of both to make sure that the Authorization Zone cannot be compromised by token leaking.

IPP Endpoints may have different trust levels. The trust level depends on the type and origin of ***IPPEndpointURI***. IPP Endpoints detected automatically with mDNS/zeroconf protocol impose additional risk over these provided by print servers or preconfigured by the system administrator.

Example: Bob is a bored programmer. One day, he noticed a new printer in his office. The printer announces itself through mDNS as “SuperPrinter 999x”. Bob quickly implements a fake IPP printer on his workstation that announces itself through mDNS as “ SuperPrinter 999x” (with a space as a first character). Confused co-workers submit their print jobs to Bob’s workstation. Bob’s fake IPP printer dumps all incoming printing jobs to a hard drive and resends them to the real printer “SuperPrinter 999x”. This new printer works perfectly and no one noticed any changes. Bob entertains himself at work by reading documents printed by his boss.

Another precaution must be taken to prevent redirection of the IPP Client to a fake Authorization Server. The IPP Client cannot trust every ***AuthorizationServerURI*** queried from an unauthorized IPP Endpoint. This issue may be solved by restricting the allowed ***AuthorizationServerURI*** to a predefined trusted list stored in the IPP Client. New ***AuthorizationServerURI***s may be added to this list by a user’s request, e.g., each time an IPP Client obtains from an IPP Endpoint an unknown ***AuthorizationServerURI***, the user is asked if a given Authorization Server can be

trusted. This way the trusted list can be extended with new ***AuthorizationServerURI***s under the user's control.

Security of connections opened by IPP Client

According to OAuth 2 specification, all communication between the IPP Client and the Authorization Server must be secured by TLS (i.e., over https). The security of the connection between an IPP Client and an Authorization Server is straightforward: the IPP Client simply verifies a certificate of the Authorization Server. These following two requirements must be met in all cases:

- ***AuthorizationServerURI*** is a global and static address (e.g., FQDN)
- Authorization Server has a valid SSL certificate

"Global" here does not mean that it must be a public address. It may be a server that is accessible only from an internal network (intranet). As a "global" address we can assume here an address that is resolvable from all locations where at least one IPP Endpoint from the Authorization Zone is resolvable.

Problems arise when we try to describe properties of an IPP Endpoint. To get some insight let's consider two extreme cases of an IPP Endpoint setup:

1. Cloud printing service. The service provides infrastructure printers as IPP Endpoints and works as a proxy between real Printers and User Devices. In this scenario, an IPP Client connects to the IPP Endpoint in the same way as to the Authorization Server. An IPP Endpoint has a unique and static URL with the same visibility as ***AuthorizationServerURI***.
2. A printer visible only in a local network (e.g. mDNS/zeroconf printer). In this case, the IPP Endpoint has a local address that is resolvable only in a particular location (e.g., only from a particular wifi network). As a result, its ***IPPEndpointURI*** may be non-unique in the Authorization Zone, and the Authorization Server may not be able to open connections to the IPP Endpoint.

In the second case, we need additional information to verify the identity of the IPP Endpoint.

Proposed modification of the original protocol

The goal of the proposed modification is to solve the security issues signaled in the previous sections and, at the same time, allow support for various types of IPP Endpoints. The final version of the use case from the original protocol may look as follows:

1. IPP Client wants to print to a given IPP Endpoint that requires OAuth authorization
2. IPP Client queries ***AuthorizationServerURI*** from the IPP Endpoint

Here, the IPP Client also validates the certificate of the IPP Endpoint. The IPP Client must not proceed if the certificate cannot be validated.

3. IPP Client registers itself to the Authorization Server (if not registered), completes through the authorization process and obtains ***AccessToken***

   The IPP Client must have the Authorization Server on its trusted list, and it must verify the server's certificate.

4. IPP Client sends to the Authorization Server the Token Exchange request to obtain ***EndpointAccessToken*** for given ***IPPEndpointURI***. ***IPPEndpointSSLFingerprint*** is added to the ***IPPEndpointURI*** as the last query parameter. The Authorization Server may use this additional information to verify the identity of printers with non-unique addresses.

5. IPP Client sends a print job to the IPP Endpoint with the obtained ***EndpointAccessToken***

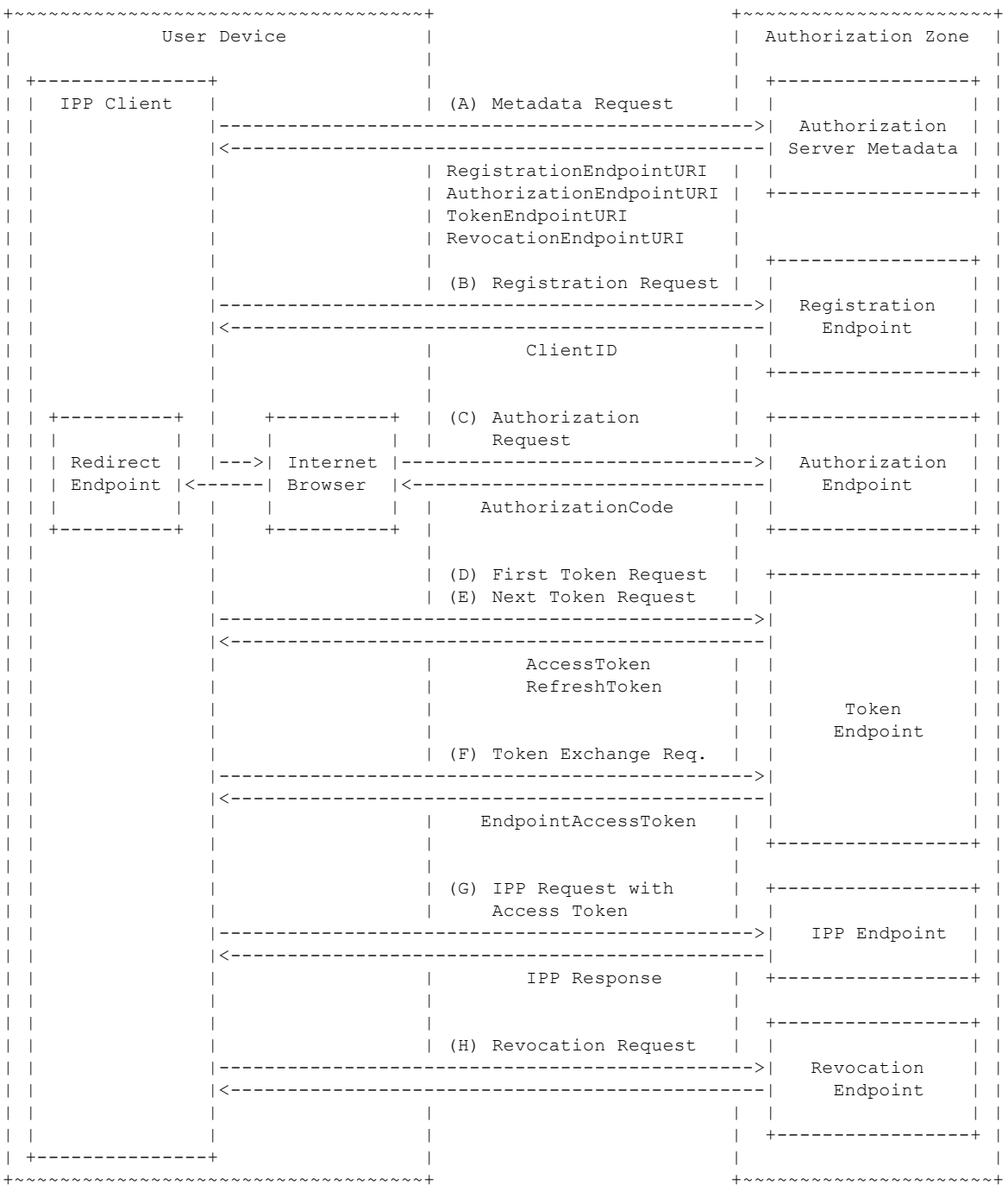
6. IPP Endpoint verifies the ***EndpointAccessToken*** and prints the document

   The communication between an IPP Endpoint and its Authorization Server is not within the scope of this document. However, we can assume that any IPP Endpoint can open a secure connection to the Authorization Server in the same way as an IPP Client.

Proposed protocol

General communication scheme

The diagram below presents the main actors and communication between them. Detailed description of requests labeled with capital letters are included in the section HTTP exchanges. Labels next to arrows pointing back to the IPP Client refer to values passed in successful responses.



Expected behavior of IPP Client

The security of the whole ecosystem is anchored in Authorization Servers. All ***AuthorizationServerURIs*** must come from trusted sources and the whole communication with them must require TLS. Each Authorization Server represents a single Authorization Zone. An IPP Client must keep track of a set of Authorization Servers it is registered to.

The IPP Client may assume that a new IPP Endpoint does not need OAuth authorization (see IPP Request Without Access Token) when communicating for the first time. However, in case of an error response indicating the necessity of OAuth authorization, the IPP Client should query the IPP Endpoint with Get-Printer-Attributes request to obtain ***AuthorizationServerURI*** stored in the attribute "oauth-authorization-server-uri" and ***Scope*** stored in the attribute "oauth-authorization-scope".

If an IPP Client is not registered to the Authorization Zone represented by a given ***AuthorizationServerURI***, it must go through the Registration Procedure, described below, and store all obtained metadata that are needed for authorization. IPP Client can register only to Authorization Zones represented by ***AuthorizationServerURI*** included in its trusted list. If a given ***AuthorizationServerURI*** is not on the trusted list, the user can be asked to add it to the list.

IPP Client holds two types of OAuth sessions:

1. Sessions created with (C) Authorization Request and (D) First Token Request are used for communication with the Authorization Servers. They are based on ***AccessToken***.
2. Sessions created with (F) Token Exchange Request. In this request the ***AccessToken*** is exchanged for ***EndpointAccessToken*** that is used in direct communication with an IPP Endpoint. These sessions are created separately for each IPP Endpoint.

All information about current OAuth sessions opened by an IPP Client must be held in volatile memory and cannot be saved to any permanent storage (to protect ***AccessToken*** and ***RefreshToken***, see RFC6749-2.1 for security of OAuth public client). During runtime, IPP Client must keep track of all currently open OAuth sessions. OAuth sessions of the first type are unambiguously identified by ***AuthorizationServerURI*** while the sessions of the second type are identified by an unique pair (***AuthorizationServerURI***, ***IPPEndpointURI***). Sessions contain the following information:

- ***AccessToken*** or ***EndpointAccessToken***
- Expiration timestamp for ***AccessToken/EndpointAccessToken***
- ***RefreshToken*** (if available)
- ***Scope*** (the value used in the (C) Authorization Request (RFC6749-4.1))

When an IPP Client tries to send a request to a printer assigned to some ***AuthorizationServerURI***, it must first check the list of current OAuth sessions and reuse an existing OAuth session when possible. If there is no session for a given pair (***AuthorizationServerURI***, ***IPPEndpointURI***) or reusing it is not possible, the IPP Client must search for a session indexed by ***AuthorizationServerURI*** and use (F) Token Exchange Request to create a missing session. If there is no OAuth session for the ***AuthorizationServerURI*** or reusing of the existing session is not possible, the IPP Client must perform the Authorization Procedure described below to open a new OAuth session.

Registration Procedure

An IPP Client executes this procedure to register itself in the new Authorization Zone with a given ***AuthorizationServerURI*** and to acquire values of required variables: ***AuthorizationEndpointURI***, ***TokenEndpointURI***, ***RevocationEndpointURI*** (optional), and ***ClientID***. The procedure consists of two steps:

1. (A) Metadata Request (RFC8414)
2. (B) Registration Request (RFC7591)

Both of these steps are described in the HTTP exchanges section.

Authorization Procedure

The goal of this procedure is to obtain a valid ***AccessToken*** for the Authorization Zone with a given ***AuthorizationServerURI***. An IPP Client must be already registered with the Authorization Zone and know the values of the following variables associated with ***AuthorizationServerURI***: ***AuthorizationEndpointURI***, ***TokenEndpointURI***, and ***ClientID*** (they are obtained in the Registration Procedure described above).

If an IPP Client does not have a corresponding ***RefreshToken***, it must execute these two steps:

1. (C) Authorization Request (RFC6749-4.1)
2. (D) First Token Request

If the IPP Client already has a ***RefreshToken*** assigned to a given ***AuthorizationServerURI***, it must try to use it to obtain a new value for an expired ***AccessToken*** instead. This step is described in the following subsection:

- (E) Next Token Request

For communication with an IPP Endpoint, the ***EndpointAccessToken*** must be obtained with the use of the request from:

- (F) Token Exchange Request (RFC8693).

All these steps mentioned above are described in the HTTP exchanges section.

Revocation Procedure

When a user session is being closed, all stored ***AccessTokens***, ***EndpointAccessTokens*** and ***RefreshTokens*** must be automatically deleted on the IPP Client side (they are supposed to be in volatile memory anyway). The session data on the server side can be deleted by sending a revocation request (see (H) Revocation Request (RFC7009)). If the AuthorizationServer supports a revocation request, the IPP Client should use it when the OAuth session with the Authorization

Zone is not needed any more. It happens when any of the following occurs:

- The IPP Client is shutting down
- ***AccessToken*** expires and there is no ***RefreshToken***
- User logged out/closed the session with Authorization Zone (if IPP Client implements this feature)
- The IPP Client may choose to close the session due to inactivity

(H) Revocation Request (RFC7009) may be called for ***RefreshTokens*** or ***AccessTokens*** (only if the corresponding ***RefreshToken*** does not exist). It must automatically invalidate all related ***EndpointAccessTokens***.

Expected behavior of IPP Endpoint

If an IPP Endpoint requires the OAuth authorization, it must have the IPP attribute “oauth-authorization-server-uri” in the Printer Description set to ***AuthorizationServerURI***. IPP Endpoints without this attribute do not require OAuth 2.0 authentication, and they do not belong to any Authorization Zone. Every IPP Endpoint can belong to at most one Authorization Zone, so the attribute “oauth-authorization-server-uri" can have only a single value. If the attribute “oauth-authorization-scope” is present its value should be used as ***Scope***. All IPP Endpoints belonging to the same Authorization Zone should have the same value of the attribute “oauth-authorization-server-uri".

The other IPP attributes that must be set appropriately are printer-uri-supported (RFC8011-5.4.1), uri-authentication-supported (RFC8011-5.4.2) and uri-security-supported (RFC8011-5.4.3). The first attribute must contain ***IPPEndpointURI***, the second one the value "oauth" and the third one the value “tls”. If supported by the IPP Endpoint, the IPP Client should use the printer-xri-supported (RFC3380-6.6) attribute instead of these three attributes.

The IPP Endpoint must not require authorization for the IPP request Get-Printer-Attributes. In the case of a missing ***EndpointAccessToken*** in the HTTP header in an incoming IPP request, the IPP Endpoint must behave as defined in the subsection IPP Request Without Access Token. When the ***EndpointAccessToken*** is present in the request, the IPP Endpoint must follow instructions from (G) IPP Request With Access Token.

HTTP exchanges

These rules must be followed by all HTTP exchanges defined in this section:

- All HTTP requests described in this section must use the https protocol. The only exception is ***RedirectURL*** that does not require encryption and the HTTP protocol.
- All communication is initiated by an IPP Client.
- All URL with parameters must be built according to rules described in RFC6749-B and conform to “application/x-www-form-urlencoded” format described in

https://www.w3.org/TR/html401/interact/forms.html#h-17.13.4. The same holds for all payloads with content of type “application/x-www-form-urlencoded”.

- Values described as ASCII strings must be within the following set: 0x20, 0x21, 0x23-0x5B, 0x5D-0x7E (all printable characters without ‘"‘ and ‘\’).

The following table summarizes all OAuth-related exchanges defined in this section:

	HTTP method	Content in request	Content in successful response	Content in error response
IPP Request Without Access Token	POST	IPP	IPP	Header: param “WWW-Authenticate: Bearer” with parameters
(G) IPP Request With Access Token (RFC6750)				
(A) Metadata Request (RFC8414)	GET	none	Payload: JSON	Payload: JSON
(B) Registration Request (RFC7591)	POST	Payload: JSON	Payload: JSON	Payload: JSON
(C) Authorization Request (RFC6749-4.1)	GET	URL parameters	URL parameters in ***RedirectURL***	URL parameters in ***RedirectURL***
(D) First Token Request (RFC6749-5)	POST	Payload: x-www-form-urlencoded	Payload: JSON	Payload: JSON
(E) Next Token Request (RFC6749-6)	POST	Payload: x-www-form-urlencoded	Payload: JSON	Payload: JSON
(F) Token Exchange Request (RFC8693)	POST	Payload: x-www-form-urlencoded	Payload: JSON	Payload: JSON
(H) Revocation Request (RFC7009)	POST	Payload: x-www-form-urlencoded	none	Payload: JSON

IPP Request Without Access Token

HTTP request

Type: POST

Header parameters:

- Content-Type: application/ipp

Payload: IPP request.

HTTP successful response

Status: 200 Ok

Header parameters:

- Content-Type: application/ipp

Payload: IPP response.

HTTP error response when Access Token is required (RFC6750-3)

Status: 401 Unauthorized

Header parameters:

- WWW-Authenticate: “Bearer” with the following parameters:
    - realm - required, user friendly name of the printing system
    - scope - optional, if set then the IPP Client must use it in Authorization Procedure

(A) Metadata Request (RFC8414)

Input: ***AuthorizationServerURI***

Output: ***AuthorizationEndpointURI***, ***TokenEndpointURI***, ***RegistrationEndpointURI***, ***RevocationEndpointURI*** (optional)

HTTP request (RFC8414-3.1)

Type: GET

URL: The URL is constructed from ***AuthorizationServerURI*** by adding the prefix “/.well-known/oauth-authorization-server” to the path component. See RFC8615 and RFC8414-3 for details.

Example: Let’s assume that ***AuthorizationServerURI***=https://my.auth.server/aaa. In this case, the metadata file is accessible as

https://my.auth.server/.well-known/oauth-authorization-server/aaa.

HTTP successful response (RFC8414-3.2)

Status: 200 OK

Header parameters:

- Content-Type: application/json

Payload: The metadata is coded in JSON format as a single object. The following fields are essential (see RFC8414-2 for full list):

- issuer - required, = ***AuthorizationServerURI***
- authorization_endpoint - required, = ***AuthorizationEndpointURI***
- token_endpoint - required, = ***TokenEndpointURI***
- registration_endpoint - required, = ***RegistrationEndpointURI***
- scopes_supported - optional, not used
- response_types_supported - required, must contain “code”
- response_modes_supported - optional, must contain “query” if specified
- grant_types_supported - optional, must contain “authorization_code” if specified
- token_endpoint_auth_methods_supported - required, must contain “none”
- revocation_endpoint - optional, = ***RevocationEndpointURI***, if it is set then IPP Client must use Revocation Procedure when the session is being closed
- revocation_endpoint_auth_methods_supported - required when revocation_endpoint is present, must contain “none”
- code_challenge_methods_supported - required, must contain “S256”

HTTP error response

Not specified, use standard HTTP statuses.

(B) Registration Request (RFC7591)

Input: ***RegistrationEndpointURI***, ***RedirectURL***

Output: ***ClientID***

HTTP request (RFC7591-3.1)

Type: POST

URL: ***RegistrationEndpointURI*** without parameters

Header parameters:

- Content-Type: application/json

- Accept: application/json

Payload: A set of parameters is encoded in a single JSON object. The following parameters are essential (see RFC7591-2 for full list):

- All parameters from ***RegistrationEndpointURI*** (these ones omitted in the URL)
- redirect_uris - required, = [***RedirectURL***, ...] (the first one is taken)
- token_endpoint_auth_method - required, an array, must contain “none”
- grant_types - optional, an array, must contain "authorization_code"
- response_types - optional, an array, must contain “code”
- client_name - optional, may be internationalized (RFC7591-2.2)
- scope - do not include it
- software_id - optional, the name (or ID) of the software (IPP Client)
- software_version - optional, the version of the software (IPP Client)
- software_statement - optional (RFC7591-2.3), JWT (RFC7519) signed with JWS (RFC7515) using a symmetric key known by the Printer Management System (but not by the IPP Client). The symmetric key must be dedicated to this IPP Client. JWT must contain at least client_name, software_id, and software_version.

HTTP successful response (RFC7591-3.2.1)

Status: 201 Created

Header parameters:

- Content-Type: application/json
- Cache-Control: no-store
- Pragma: no-cache

Payload: The single JSON object with the following parameters:

- client_id - required, = ***ClientID***
- client_id_issued_at - optional, time at which the client identifier was issued represented as the number of seconds from 1970-01-01T00:00:00Z in UTC
- All fields assigned to the registered client on the server side. It must include all fields specified in the request, and their values must be the same as these specified in the request.

HTTP response for OAuth 2.0 error (RFC7591-3.2.2)

Status: 400 Bad Request

Header parameters:

- Content-Type: application/json
- Cache-Control: no-store

- Pragma: no-cache

Payload: A single JSON object with the following fields:

- error - required, single ASCII error code string, one of the following:
  - invalid_redirect_uri - The value of one or more redirection URIs is invalid.
  - invalid_client_metadata - The value of one of the client metadata fields is invalid and the server has rejected this request.
  - invalid_software_statement - The software statement presented is invalid.
  - unapproved_software_statement - The software statement presented is not approved for use by this authorization server.
- error_description - optional, human-readable ASCII text description of the error used for debugging.

(C) Authorization Request (RFC6749-4.1)

Input: ***AuthorizationEndpointURI***, ***ClientID***, ***RedirectURL***, ***Scope***

Output: ***AuthorizationCode***

This request is not sent directly from the IPP Client. Instead, the IPP Client opens an available internet browser (called in OAuth 2.0 specification as “user agent”) and feeds it with the URL. The internet browser sends an HTTP GET request to a given URL and allows the user to go through the whole authorization process. If the access is granted, the flow is returned to the IPP Client by sending to the Internet Browser the HTTP 302 Found response that is redirected to the IPP Client. Possible implementations of this stage across different operating systems are described in RFC8252-7**.**

First HTTP request from the internet browser (RFC6749-4.1.1)

Type: GET

URL: ***AuthorizationEndpointURI*** extended by adding the following parameters:

- response_type - required, = “code”
- response_mode - optional, if specified then = ”query”
- client_id - required, = ***ClientID***
- redirect_uri - required, = ***RedirectURL***
- scope - optional, = ***Scope*** (scopes defined in IPP Endpoint, set if ***Scope*** is specified)
- state - required, an opaque value used by the client to maintain state between the request and callback
- code_challenge - required, created according to RFC7636-4 with the algorithm “S256”
- code_challenge_method - required, = ”S256” (RFC7636-4.3)

Last HTTP response to the internet browser in case of success (RFC6749-4.1.2)

Status: 302 Found

Header parameters:

- Location: ***RedirectURL*** with the following parameters:
  - code - required, = ***AuthorizationCode***
  - state - required, = the exact value received in the first HTTP request

Last HTTP response to the internet browser in case of error (RFC6749-4.1.2.1)

Status: 302 Found

Header parameters:

- Location: ***RedirectURL*** with the following parameters:
  - error - required, a single ASCII error code from the following:
    - invalid_request - the request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.
    - unauthorized_client - the client is not authorized to request an authorization code using this method.
    - access_denied - the resource owner or authorization server denied the request.
    - unsupported_response_type - the authorization server does not support obtaining an authorization code using this method.
    - invalid_scope - the requested scope is invalid, unknown, or malformed.
    - server_error - the authorization server encountered an unexpected condition that prevented it from fulfilling the request. This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect.
    - temporarily_unavailable - the authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server. This error code is needed because a 503 Service Unavailable HTTP status code cannot be returned to the client via an HTTP redirect.
  - error_description - optional, human-readable ASCII text providing additional information, used to assist the client developer in understanding the error that occurred.
  - error_uri - optional, a URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. It must be an ASCII string without a 0x20 character.
  - state - required, = the exact value received in the first HTTP request.

(D) First Token Request (RFC6749-5)

Input: ***TokenEndpointURI***, ***ClientID***, ***RedirectURL***, ***AuthorizationCode***

Output: ***AccessToken***, ***RefreshToken***

HTTP request (RFC6749-4.1.3)

Type: POST

URL: ***TokenEndpointURI*** without parameters

Header parameters:

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

Payload: A set of parameters saved as “application/x-www-form-urlencoded” and encoded in “UTF-8”. It must contain the following parameters:

- All parameters from ***TokenEndpointURI***
- grant_type - required, = ”authorization_code”
- code - required, = ***AuthorizationCode***
- redirect_uri - required, = ***RedirectURL***
- client_id - required, = ***ClientID***
- code_verifier - required, =*code verifier*, see RFC7636-4.5

HTTP successful response (RFC6749-4.1.4)

Status: 200 Ok

Header parameters:

- Content-Type: application/json;charset=UTF-8
- Cache-Control: no-store
- Pragma: no-cache

Payload: The content is a JSON with the following values (RFC6749-5.1):

- access_token - required, = ***AccessToken***
- token_type - required, = "bearer"
- expires_in - optional, the lifetime in seconds of the access token.
- refresh_token - optional, = ***RefreshToken***
- scope - optional, not used

HTTP response for OAuth 2.0 error (RFC6749-5.2)

Status: 400 Bad Request

Header parameters:

- Content-Type: application/json;charset=UTF-8
- Cache-Control: no-store
- Pragma: no-cache

Payload: A single JSON object with the following fields:

- error - required, single ASCII error code string, one of the following:
    - invalid_request - The request is missing a required parameter. It includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, uses more than one mechanism for authenticating the client, or is otherwise malformed.
    - invalid_client - Client authentication failed (e.g., unknown client, no client authentication included, or unsupported authentication method).
    - invalid_grant - The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, does not match the redirection URI used in the authorization request, or was issued to another client.
    - unauthorized_client - The authenticated client is not authorized to use this authorization grant type.
    - unsupported_grant_type - The authorization grant type is not supported by the authorization server.
    - invalid_scope - The requested scope is invalid, unknown, malformed, or exceeds the scope granted by the resource owner.
- error_description - optional, human-readable ASCII text description of the error used for debugging.
- error_uri - optional, a URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. It must be an ASCII string without a 0x20 character.

(E) Next Token Request (RFC6749-6)

Input: ***TokenEndpointURI***, ***RefreshToken***

Output: ***AccessToken***, ***RefreshToken***

HTTP request

Type: POST

URL: ***TokenEndpointURI*** without parameters

Header parameters:

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

Payload: A set of parameters saved as “application/x-www-form-urlencoded” and encoded in “UTF-8”. It must contain the following parameters:

- All parameters from ***TokenEndpointURI***
- grant_type - required, = ”refresh_token”
- refresh_token - required, = ***RefreshToken***
- scope - do not include it

HTTP response

Responses are the same as described in “(D) First Token Request”. The server may issue a new ***RefreshToken***, in which case the client must discard the old ***RefreshToken*** and replace it with a new one.

(F) Token Exchange Request (RFC8693)

Input: ***TokenEndpointURI***, ***AccessToken***, ***IPPEndpointURI***

Output: ***EndpointAccessToken***

HTTP request (RFC8693-2.1)

Type: POST

URL: ***TokenEndpointURI*** without parameters

Header parameters:

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

Payload: A set of parameters saved as “application/x-www-form-urlencoded” and encoded in “UTF-8”. It must contain the following parameters:

- All parameters from ***TokenEndpointURI***
- grant_type - required, = ”urn:ietf:params:oauth:grant-type:token-exchange”
- resource - required, = ***IPPEndpointURI*** with additional query parameter SSLFingerprint= ***IPPEndpointSSLFingerprint***
- scope - do not include it
- subject_token - required, = ***AccessToken***
- subject_token_type - required, = “urn:ietf:params:oauth:token-type:access_token”

HTTP successful response (RFC8693-2.2)

Status: 200 Ok

Header parameters:

- Content-Type: application/json
- Cache-Control: no-store
- Pragma: no-cache

Payload: The content is a JSON with the following values (RFC6749-5.1):

- access_token - required, = ***EndpointAccessToken***
- issued_token_type - required, but not used, if unsure set it to "urn:ietf:params:oauth:token-type:access_token"
- token_type - required, = "bearer"
- expires_in - optional, the lifetime in seconds of the access token.
- refresh_token - do not use it
- scope - do not use it

HTTP response for OAuth 2.0 error (RFC6749-5.2)

Status: 400 Bad Request

Header parameters:

- Content-Type: application/json
- Cache-Control: no-store
- Pragma: no-cache

Payload: A single JSON object with the following fields:

- error - required, single ASCII error code string, one of the following:
    - invalid_request - The request is missing a required parameter. It includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, uses more than one mechanism for authenticating the client, or is otherwise malformed.
    - invalid_grant - The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, does not match the redirection URI used in the authorization request, or was issued to another client.
    - unauthorized_client - The authenticated client is not authorized to use this authorization grant type.
    - unsupported_grant_type - The authorization grant type is not supported by the authorization server.
    - invalid_scope - The requested scope is invalid, unknown, malformed, or exceeds the scope granted by the resource owner.
    - invalid_target - The URI passed in the resource parameter is unknown or incorrect.
- error_description - optional, human-readable ASCII text description of the error used for

debugging.
- error_uri - optional, a URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. It must be an ASCII string without a 0x20 character.

(G) IPP Request With Access Token (RFC6750)

Input: ***AccessToken***

HTTP request (RFC6750-2.1)

Type: POST

Header parameters:

- Content-Type: application/ipp
- Authorization: Bearer ***AccessToken***

Payload: IPP request.

HTTP successful response

Status: 200 Ok

Header parameters:

- Content-Type: application/ipp

Payload: IPP response.

HTTP error response (RFC6750-3)

Status: 400 Bad Request or 401 Unauthorized or 403 Forbidden

Header parameters:

- WWW-Authenticate: Bearer with the following parameters:
  - realm - required, user friendly name of the printing system
  - scope - optional, a required ***Scope*** value
  - error - optional, one of the following (RFC6750-3.1):
    - invalid_request - The request is missing a required parameter. It includes an unsupported parameter or parameter value, repeats the same parameter, uses more than one method for including an access token, or is otherwise malformed. It may be returned only for the HTTP 400 (Bad Request) status code.
    - invalid_token - The access token provided is expired, revoked, malformed, or invalid for other reasons. It may be returned only for the HTTP 401

(Unauthorized) status code. The client may request a new access token and retry the protected resource request.
        - insufficient_scope - The request requires higher privileges than provided by the access token. It may be returned only for the HTTP 403 (Forbidden) status code
    - error_description - optional
    - error_uri - optional

(H) Revocation Request (RFC7009)

Input: ***RevocationEndpointURI***, ***RefreshToken*** or ***AccessToken***

The exchange is considered successful when the server revoked a given token or when the token is not known by the server. It is so, because the server does not have to store or track expired tokens and revoking expired tokens cannot cause error response.

If the server responds with HTTP status code 503, the client must assume the token still exists and may retry after a reasonable delay. The server may include a "Retry-After" header in the response to indicate how long the service is expected to be unavailable to the requesting client.

HTTP request (RFC7009-2.1)

Type: POST

URL: ***RevocationEndpointURI*** without parameters

Header parameters:

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

Payload: A set of the following parameters saved as “application/x-www-form-urlencoded”:

- All parameters from ***RevocationEndpointURI***
- token - required, = ***RefreshToken*** if defined, otherwise ***AccessToken***

HTTP successful response (RFC7009-2.2)

Status: 200 Ok

HTTP error response (RFC7009-2.2.1)

Status: 400 Bad Request

Header parameters:

- Content-Type: application/json;charset=UTF-8
- Cache-Control: no-store

- Pragma: no-cache

Payload: A single JSON object with the following fields:

- error - required, single ASCII error code string, one of the following:
    - unsupported_token_type - The authorization server does not support the revocation of the presented token type. That is, the client tried to revoke an access token on a server not supporting this feature.

Others

Querying the list of available IPP Endpoint from IPP System

The standardized API for accessing IPP Systems is defined in the document IPP System Service v1.0. We are interested in the request Get-Printers described in the section 6.1.4. It allows the IPP Client to obtain ***IPPEndpointURI***s via the attributes *printer-xri-supported*. However, the document does not mention the attribute *oauth-authorization-server-uri*. This probably means that it may be or may be not present. In any case, its value must be queried directly from the IPP Endpoint via the IPP request Get-Printer-Attribute.

The access to the IPP System is controlled in a similar way as an access to IPP Endpoints. The client must first send an IPP request Get-System-Attributes to obtain the attribute *oauth-authorization-server-uri* for the IPP System. After successful authorization, the client is able to query the list of available printers with the request Get-Printers. This procedure is proposed in the errata filed for the document IPP System Service v1.0 (it can be found at https://www.pwg.org/dynamo/issues.php).

In general, IPP System endpoints can be treated in analogical way as IPP Endpoints.