Web Authorization Protocol
Internet-Draft
Intended status: Experimental
Expires: 1 September 2026

Y. Song
L. Li
Huawei
Y. Jiang
F. Liu
Huawei Singapore

28 February 2026

OAuth2.0 Extension for Multi-AI Agent Collaboration
draft-song-oauth-ai-agent-collaborate-authz-01

Abstract

This draft proposes an authorization method for task-oriented multi-AI agent collaboration scenarios, where a leading agent coordinates sub-agents to complete complex tasks. The methods extends the OAuth 2.0 protocol by adding fields in token and message flows, enabling sub-agents execute the task as a group. This approach greatly simplifies the authorization process for a task group, avoids efficiency loss from repeated interactions between multiple sub-AI agents and the authorization server, meanwhile maintaining full compatibility with existing OAuth 2.0 workflows.

Requirements Language

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 [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 1 September 2026.

Table of Contents

1. Introduction

AI agents are capable of handling tasks. They can integrate multiple technologies such as natural language understanding, data analysis, and logical reasoning to meet multi-step and cross-scenario requirements. However, a single AI agent has a limited knowledge scope and restricted functions, making it difficult to independently complete complex tasks that span multiple professional domains and require different capabilities.

Collaboration among a group of AI agents is an essential approach for such complex tasks. Taking the demand for "real-time health advice" as an example, this intent can be split into three tasks: "collect the user’s health data", "predict the user’s health status" and "provide health advice based on health status". These tasks need to be accomplished by corresponding professional sub-AI agents: the data collection agent is responsible for privacy-compliant data collection, the health status prediction agent invokes medical models for analysis, and the advice generation agent outputs solutions based on health guidelines.

What’s more, accordingto some existing research[multi-agent-research-system], a leading agent is necessary for coordination. In the above example, the leading agent can receive the intent "give me real-time health advice" from user, understand it and split it into tasks. Additionally, the leading agent selects sub-agents capable of fulfilling the tasks and distributes these tasks to them. These sub-agents can be referred to as a task group. The sub-agents can be refered as task group. Finally, the leading agent integrates the results returned by the sub-AI agents to form complete health advice. Without the leading agent, task execution is prone to confusion and final results are difficult to unify.

There are two typical policies for the leading agent to select sub-agents and form a task group: 1) The static policy determines all tasks and sub-agents upfront before execution, enabling predictable scheduling and stable group composition; 2) The dynamic policy incrementally identifies tasks and selects sub-agents during execution, allowing real-time adjustment based on intermediate results and improving adaptability to changing conditions.

In such collaborative scenarios, if each sub-AI agent in a task group applies for permissions from the authorization server individually, the authorization server must verify the identity and permissions of each sub-agent one by one and generate access tokens for them. This leads to problems such as frequent interactions, high computing resource consumption on the authorization server, and low efficiency in the token issuance process.

Therefore, this draft proposes two efficient authorization methods: 1) The leading agent communicates with the authorization server centrally and applies for access tokens on behalf of all sub-agents in the task group before task execution begins; 2) The leading agent applies for an access token for a specific task and issues task credentials bound to the access token to the corresponding sub-agents. The first method is suitable for the static policy, while the second method is suitable for the dynamic policy. Both methods can greatly simplify the authorization process for sub-AI agents and avoid efficiency loss caused by repeated interactions.

2. Terminology

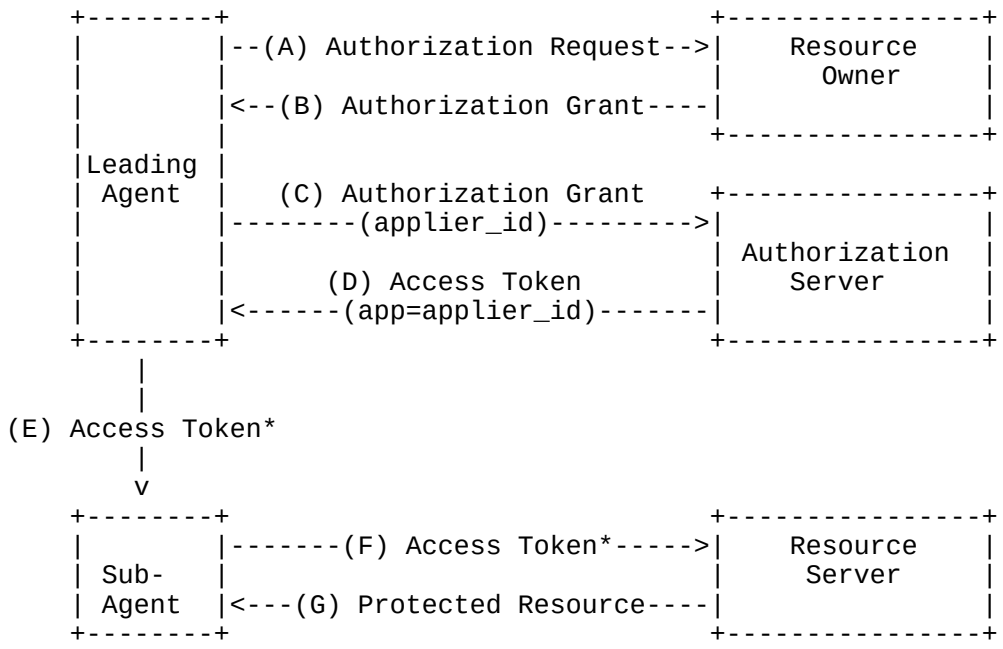
Applier: a role that request the access token on be half of client(s), e.g. a leading AI agent.

DPoA: Demonstration Proof of Attribute.

This draft uses the terms "authorization server", "client", "resource server" defined by OAuth 2.0 [RFC6749]

3. Procedure of Static Task Group Authorization

This protocol flow is modeled after [RFC6749] and extends it with a *applier_id* field to enable authorization request by the applier.



3.1. Extension for Client Registration

Before initiating the protocol, the leading agent registers with the authorization server, as defined in [RFC6749].

In this draft, the leading agent’s role or capability is incorporated into its registration process. For example, the leading agent’s profile includes a "capability" parameter with the value "resolve intent and distribute tasks".

3.2. Authorization Request & Response

This section refers to step (A) and (B).

The leading agent receives a requirement from the user, resolves it and determines the task(s) and sub-agent(s) needed to fulfill the requirement. For example, when the user sends the intent "give me real-time health advice" to the leading agent, the leading agent resolves this intent and identifies three tasks: task 1 (collect the user’s health data), task 2 (predict the user’s health status) and task 3 (provide health advice based on health status). The leading agent then utilizes the agent discovery process to select sub-agent(s) capable of performing the respective tasks. For instance, task 1 assigned to sub-agent 1, task 2 to sub-agent 2, and task 3 to

sub-agent 3. In the subsequent authorization process proposed in this draft, the leading agent is designated as the applier, and the selected sub-agent(s) are designated as the client(s).

The leading agent requests authorization from the resource owner, as defined in [RFC6749].

3.3. Access Token Request & Response

This section refers to step (C) and D).

The leading agent requests an access token by authenticating with the authorization server and presenting the authorization grant, as defined in [RFC6749]. The message may also include the following parameter:

applier_id

OPTIONAL. The identifier of the applier which is requesting access token on be half of the client(s).

The authorization server authenticates the leading agent and validates that the leading agent is capable of requesting access tokens on behalf of sub-agent(s). For example, it checks whether the leading agent’s profile includes a "capability" parameter whose value includes "distribute tasks".

If the validation is successful, the authorization server may verify grants as defined in [RFC6749]. Then the authorization server generates access token and send access token response message to the leading agent. The access token is similar to OAuth2.0, except that it consists of an additional claim to specify the applier.

app

OPTIONAL. The identifier of the applier which is requesting access token on be half of the client(s).

The access token includes one *app* parameter and multiple *sbj*-*aud*-*scope* pairs. Below is an example:

```
{

"iss": "authorization server ID",

   "app": "leading agent ID",

   "sbj": "sub AI agent1 ID",
```

```
        "aud": "resource server 1 ID",

        "sbj": "sub AI agent 2 ID",

        "aud": "resource server 1 ID", "resource server 2 ID",

        "sbj": "sub AI agent 3 ID",

        "aud": "resource server 2 ID", "resource server 3 ID"

        ...

      }
```

If the validation fails, the access token response message may include the reason for failure. [RFC6749] has defined types of error response. In this case, the authorization server may use a new error message "unauthorized_applier" to indicate that the leading agent is not capable of requesting access tokens on be half of sub-agent(s).

3.4. Access Token Transmit

This section refers to step (E).

The leading agent sends the task ID and access token* to the sub-agent(s). Based on local policies and regulations, the leading agent may use privacy protection mechanisms to process the access token. Thus, the access token* may be the same as the access token, or generated from the access token using privacy protection mechanisms. For example, the leading agent uses a selective disclosure algorithm to generate access token 1, access token 2, and access token 3 from the access token received in step (D). Access token 1 may include the leading agent ID and sub-agent 1 ID; access token2 may include the leading agent ID and sub-agent 2 ID; access token3 may include the leading agent ID and sub-agent 3 ID. The applier may then send access token 1 to sub-agent 1, access token 2 to sub-agent 2, and access token 3 to sub-agent 3.

The sub-agent(s) validate the proof of the received access token* and check whether the applier ID refers to a trustworthy leading agent. For example, each AI agent is preconfigured with a list of trusted AI agents, and any one of the trusted AI agents that acts as a leading agent will pass the validation.

If the validation fails, the client(s) may send a response message to the applier with a failure indication of "unknown_applier".

3.5. Access Token Verify

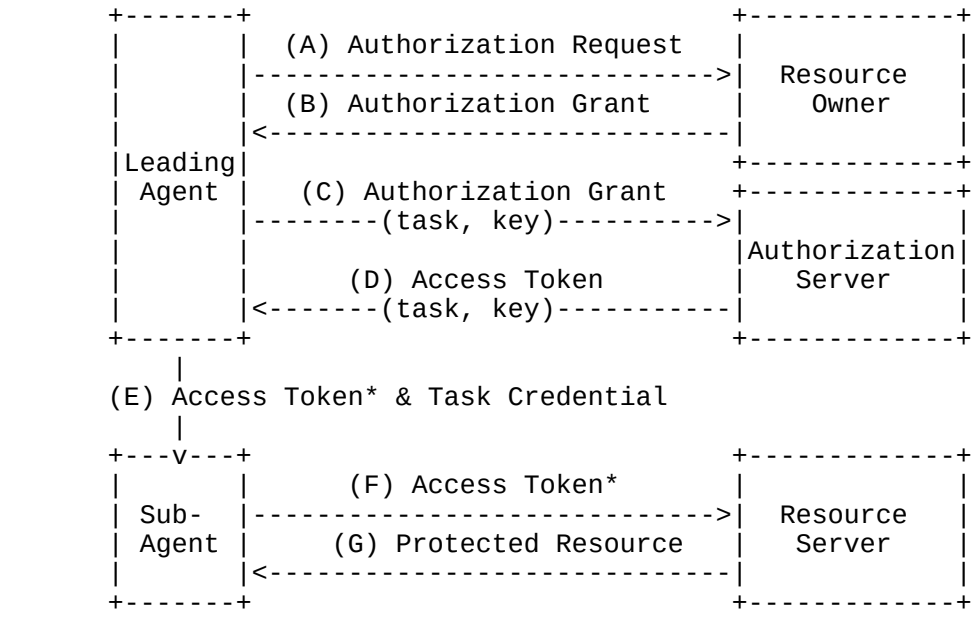
This section refers to step (F) and (G).

The sub-agent(s) execute the task according to message received in step (E). When resources are needed, the sub-agent(s) request the protected resource from the resource server and present the access token*, as defined in [RFC6749]. The resource server validates the access token*, as defined in [RFC6749]. If the access token* includes multiple *sbj*-*aud* pairs, the resource server may verify that its own ID and the sub-agent's ID are in the same pair. In previous example of the access token, "sub-agent 1 ID"and "resource server 1 ID" are in the same pair.

If the validation succeeds, the resource server may provide the resource to the client(s).

4. Procedure of Dynamic Task Group Authorization

This protocol flow is modeled after [RFC6749] and extends it with task credentials bound with the access token.



4.1. Extension for Client Registration

Before initiating the protocol, the leading agent registers with the authorization server.

4.2. Authorization Request & Response

This section refers to step (A) and (B).

The leading agent receives a requirement from the user, resolves it and determines the task(s) and client(s) needed to fulfill the requirement. The leading agent requests authorization from the resource owner. Detailed prcedures are similar to section 3.2.

4.3. Access Token Request & Response

This section refers to step (C) and D).

The leading agent requests an access token by authenticating with the authorization server and presenting the authorization grant, as defined in [RFC6749]. The message may also include the following parameter:

task

OPTIONAL. The identifier or description of the task which is resolved from intent.

key

OPTIONAL. The index that point to the public key of the leading agent that is used to issue task credentials.

The authorization server authenticates the leading agent and validates that the leading agent is capable of executing specific task. What’s more, the authorization server validated that the leading agent holds the private key related to the *key* received in the request message.

If the validation is successful, the authorization server may verify grants as defined in [RFC6749]. Then the authorization server generates access token and send access token response message to the applier. The access token is similar to OAuth2.0 and OAuth2.0 DPoP, reuse the *subject* parameter, and add a new parameter to specify the task and related public key. This access token can be named as DPoA token.

att

OPTIONAL. The attribute of the sub-agent that is required to

permit the sub-agent to request resources indicated in the access token. In this case, this parameter is set to the indication of public key of the leading agent that is used to issue task credentials.

Below is an example:

{

"iss": authorization server ID,

"sbj": task ID,

"aud": resource server 1 ID,

"att": public key for specific task

...

}

If the validation fails, the access token response message may include the reason for failure. In this case, the authorization server may use a new error message "unrecognized_pk" to indicate that the validation of *key* sent by the leading agent is wrong.

4.4. Access Token Transmit

This section refers to step (E).

After the leading agent selects sub-agent to execute task, it generates task credential and sends the credential with access token* to the sub-agent(s). The task credential is used to prove that the subject of the credential is selected by the leading agent to execute specific task. Optionally it includes the information of the related access token. The credential is signed by the leading agent using the private key related to the public key indicated by the *att* parameter in the access token. This credential can have multiple formats. Below is an example:

{

"issuer": leading agent ID,

"subject": sub-agent ID,

"attribute": task ID

```
        "related token": hash of the access token

        ...

        "proof": signature of leading agent

      }
```

The leading agent may resolve a list of multiple tasks based on one intent from the user. In this case, the *sbj* in the access token may involve several tasks. Based on local policies and regulations, the leading agent may use privacy protection mechanisms to process the access token. For example, the leading agent uses a selective disclosure algorithm to generate access token 1, access token 2, and access token 3 from the access token received in step (D). Access token 1 may include task ID 1; access token2 may include ttask ID 2; access token3 may include task ID 3. The leading agent may then send access token 1 to sub-agent 1, access token 2 to sub-agent 2, and access token 3 to sub-agent 3.

The sub-agent(s) validate the the received access token* and task credential, and validate the relationship of them.For example, to validate the task credenial, the sub-agent(s) check whether the *subject* indicates a trusted leading agent, and whether the *proof* value is right; to validate the relationship, the sub-agent(s) check whether the task ID in the access token* is the same as the task credential, whether the key indicated by the access token* can be used to validate the proof of the task credential, and whether the token hash in the task credential is same as the hash of the access token*.

If the task credential validation fails, the sub-agent(s) may send a response message to the applier with a failure indication of "invalid_credential". If the relationship validation fails, the sub-agent(s) may send a response message to the applier with a failure indication of "unknown_credential".

4.5. Access Token Verify

This section refers to step (F) and (G).

The sub-agent(s) execute the task according to message received in step (E). When resources are needed, the sub-agent(s) request the protected resource from the resource server and present the access token*, as defined in[RFC6749]. Meanwhile, the sub-agent(s) send the task credential to the resource server, The resource server validates the access token*, as defined in[RFC6749]. And validates the task credential and the relationship between the access token* and the task credential.

If the validation succeeds, the resource server may provide the resource to the client(s).

5. Security Considerations

TBD

6. References

6.1. References

6.2. Normative References

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

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

6.3. Informative References

[multi-agent-research-system] Anthropic, "How we built our multi-agent research system", 13 June 2025.

Authors' Addresses

Yurong Song
Huawei
Email: songyurong1@huawei.com

Lun Li
Huawei
Email: lilun20@huawei.com

Yuning Jiang
Huawei Singapore
Email: jiangyuning2=40h-partners.com@dmarc.ietf.org

Faye Liu
Huawei Singapore
Email: liufei19@huawei.com