Security Working Group

Cascaded Authorization Service

Draft v0.9

Mohammad Jafari, Kathleen Connor

January 15, 2018

Table of Contents

List of Figures

1 INTRODUCTION

This document introduces the concept of Cascaded Authorization in and discusses the details of the design and flows. Cascaded authorization enables systems using OAuth 2.0 and its User-Managed Access (UMA) profile, to set up multiple Authorization Servers each dedicated to handling one type of authorization policy. This is an extension of the plain OAuth 2.0 and UMA in which a single Authorization Server makes the ultimate decision to grant or reject an access request.

While our main focus and demo implementation of this concept has been healthcare-specific and particularly focused on the case of enforcing overarching and patient consent privacy policies using two separate authorization servers in exchange scenarios, this document discusses the concepts and the flow in more abstract and broad terms without using the healthcare-specific use-case nomenclature. This will provide the opportunity for others in the authorization community, especially the community of the User-Managed Access (UMA) profile of OAuth 2.0 to find other use-cases that can benefit from this architecture.

2 CONCEPT

The idea of having an Authorization Service originates from the need to gather all authorization responsibilities that were often scattered in different sub-systems into a single place so that policies can be configured at a single central point. Central to most authorization architectures is, thus, the idea of a single Policy Decision Point (PDP) which is in charge of processing and understanding all authorization policies, and accordingly, responds to queries about whether an access request is authorized.

As authorization systems get more complicated and data owners are given more control over the policies governing access to their data, more often than not, a data element is subject to various policies set by different authorities. So, the assumption of a single point of policy decision is becoming less realistic and less scalable.

In healthcare exchange, for example, collected data is subject to overarching policies from federal and state laws as well as specific organizational policies and patient's own privacy preferences.

Considering the policies set by the data owner/subject (such as patient consents) brings in a large number of policies into the authorization system which introduces a whole set of new challenges. Firstly, there are usually a large number of such policy makers and it can increasingly become a burden to capture, maintain, process, and manage such a large number of policies and develop the suitable user interfaces for enabling these users to author and configure their policies. This motivates outsourcing this service to a third-party authorization service which takes care of all such tasks and provides the answer to authorization queries —as far as those policies are concerned. For example, a third-party Patient Consent Authorization service can handle all the responsibilities to collect, maintain, update, and analyze patient consents, and simply responds to the queries whether (and to what extent) a requesting party is authorized to access a patient's record.

Secondly, individual data owners' policies, such as patient consents, may often include sensitive information or clues that could be indicative of sensitive information. So, it is desirable from a separation-of-duties viewpoint, to have separate decision points processing different policies and not allow a single Authorization Server to access to all applicable policies.

The crux of the Cascaded Authorization proposal is to add the capability to consult more than one Authorization Server in an UMA/OAuth architecture. In this architecture, multiple Authorization Servers can exist, each in charge of processing and handling one group of policy and making authorization decisions based on them.

Upon receiving an access request from a requesting party, a Principal Authorization Server determines which one of the Authorization Servers need to express their decisions regarding that request and sends the Authorization Request to them. The principal Authorization Server then collects and combines all of these decisions and make the ultimate call about authorizing the request.

3 ACTORS

In a typical OAuth/UMA flow, there are three actors:

- **Resource Server (RS)**: a server which stores protected resources and is capable of providing access to them.
- **Client**: the party which intends to access some resource in the care of the RS. We sometimes refer to the client as Requester, Requesting Party, or Recipient.
- **Authorization Server (AS)**: the server in charge of making authorization decisions. The AS is capable of:
    - evaluating authorization requests according to some authorization policies,
    - determining whether and to what extent access can be allowed,
    - issuing access tokens to requesters (called the Requesting Party Token (RPT) in UMA), and
    - responding to verification requests about issued tokens.

In Cascaded Authorization, there can be more than one authorization server: A Principal AS which is the final adjudicator of an access request, and one or more Secondary Authorization Servers each of which is in charge of handling a certain category of authorization policies:

- **Principal AS**: The principal AS is in immediate connection with the RS and its approval —in the form of issuing a token— is required for every single access request received by the RS. Depending on policies and configurations, this Principal AS may decide to require further approvals from one or more Secondary Authorization Servers as a precondition to issuing the final access token. The Principal AS will collect all such decisions, adjudicate them using a decision-combining policy, and eventually decide whether or not to issue an access token.
- **Secondary AS**: A secondary AS is an OAuth/UMA server in charge of handling a certain type or group of policies in its care, for example, patient consents. The approval of a Secondary AS denotes that access is authorized as far as that particular type of policies are concerned. The Secondary AS communicates this decision by issuing an Access Token. Whether or not the decision of a Secondary AS is necessary and whether or not its decision will prevail depends on the decision of the Principal AS.

Based on its configured policies, the Principal AS decide whether to require a decision from one or more Secondary Authorization Servers, and adjudicates them based on some combining policy, such as simple AND/OR, majority vote, or more sophisticated combining logic for combining the scopes granted by each Secondary AS[1].

This architecture provides a powerful distributed mechanism for separating the authorization decision-making for different classes of policy amongst a number of Secondary Authorization Servers and removes the need to have a single omniscient authorization server aware of all the applicable policies. In use-cases where there is a large number of policies to be analyzed, such as the case of patient consent, this architecture enables distribution of decisions making and separation of duties.

As an example, consider the healthcare exchange use-case where various types policies ranging from patient consent to jurisdictional policies must be followed. In this case, the Principal AS is configured to analyze and enforce the overarching and organizational policies while handling of the patient consent is delegated to Secondary Authorization Servers. The Principal AS in this case decides, according to the overarching policies and other configurations, which requests depend on the patient's consent, in which case it refers the client to the Secondary AS, i.e. a Patient Consent AS, to acquire authorization. For example, exchange of information between the VA and DoD does not require the patient consent. In this case, the primary AS, simply issues the Access Token for the requester. When a third-party organization, such a hospital requests access, however, the principal AS determines, based on the overarching policies, that this exchange requires the patient consent, so it redirects the requester to the Patient Consent AS to acquire an Access Token from that AS first.

In this document, we only discuss the case where there is a single Secondary AS; extending the design to involve more than one Secondary AS is straightforward.

4 OVERVIEW

Figure 1 depicts a high-level overview of the flow.

[1] To clarify a technical detail, we are aware that if any of the Secondary Authorization Servers refuse to issue an RPT altogether (i.e. an HTTP 403 response), then the flow halts and this means evaluation of a disjunctive (OR) combining cannot move forward. However, this can be fixed by having the Secondary AS issue an RPT with an empty scope/permission set to indicate the intention to deny an access request.

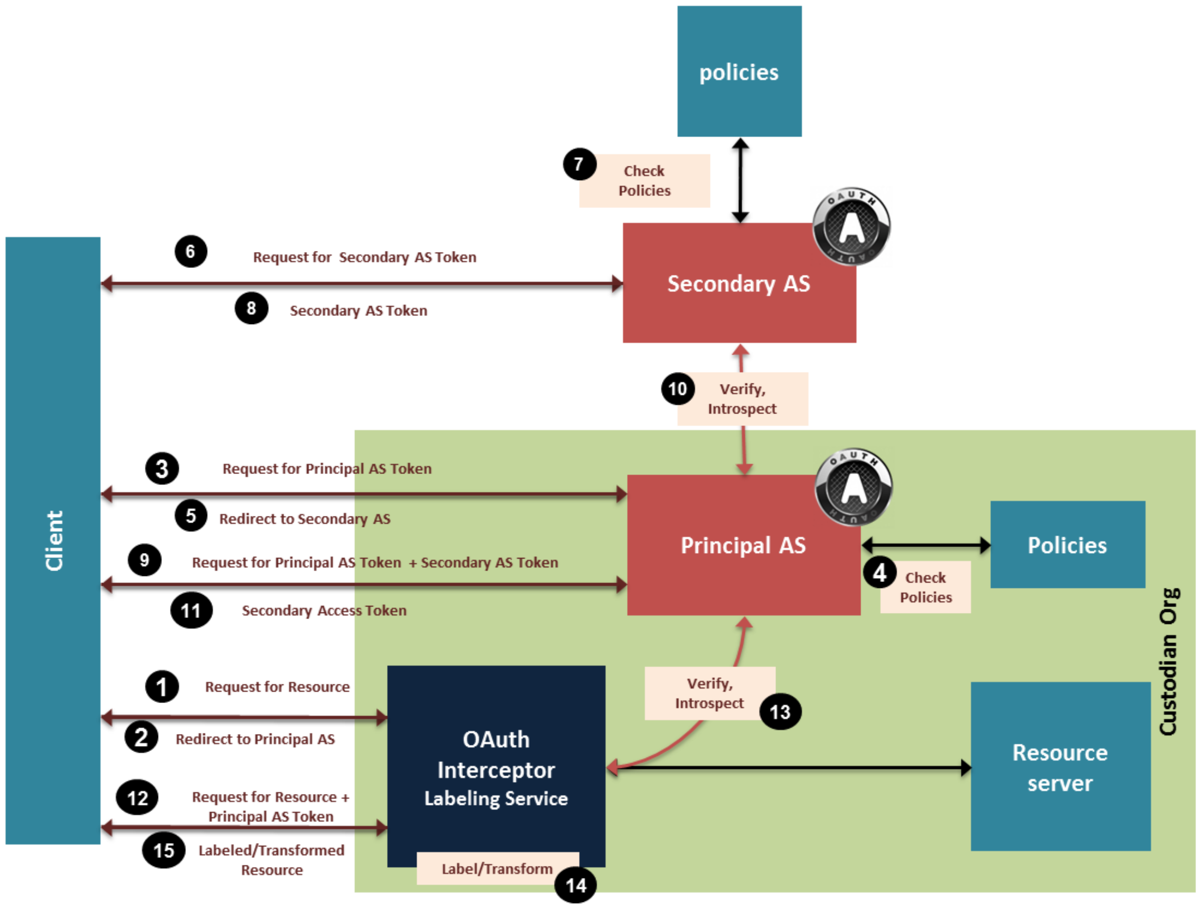
policies
7 Check Policies
Secondary AS
6 Request for Secondary AS Token
8 Secondary AS Token
10 Verify, Introspect
Client
3 Request for Principal AS Token
5 Redirect to Secondary AS
9 Request for Principal AS Token + Secondary AS Token
11 Secondary Access Token
Principal AS
Policies
4 Check Policies
1 Request for Resource
2 Redirect to Principal AS
12 Request for Resource + Principal AS Token
15 Labeled/Transformed Resource
OAuth Interceptor Labeling Service
Verify, Introspect 13
Resource server
Custodian Org
Label/Transform 14

Figure 1. High-level flow of cascaded authorization

5 ASSUMPTIONS AND PRECONDITIONS

There are a number of assumptions and preconditions in this flow.

5.1 RESOURCE

We assume that the resources are already stored at the RS and the Client knows about this and can construct a suitable query to request the resources it needs from the RS.

5.2 PROVISIONING

We assume that the Client knows where the RS resides, and already has the credentials required to speak to all of the Authorization Servers. This may require prior handshaking and provisioning to introduce the client as a legitimate communicator with each of the Authorization Servers. In a similar vein, the RS needs to have been introduced to the Principal AS, and similarly, the Principal AS needs to be introduced to all the Secondary Authorization Servers.

5.3 NAMING CONVENTIONS

The RS and all the Authorization Servers need to establish a mechanism to be able to determine which policies at which server applies to which resource. This may require a common naming convention to refer to resource sets. For example, in the case of healthcare exchange, the RS and the Secondary (Consent) AS need to have a reliable mechanism to communicate a patient ID in order to determine which patient's consent applies to a given request.

5.4 POLICY SETUP

We also assume that policies are already set up and maintained by the respective administrators, or other authorized policymakers at each of the Authorization Servers.

6 DETAILED FLOW

Figure 1 depicts the detailed sequence diagram of a successful cascaded authorization flow. Details are discussed below.

6.1 CLIENT'S FIRST REQUEST TO RESOURCE SERVER WITHOUT ACCESS TOKEN

The first time that client contacts the RS, the request does not include an Access Token. Upon receiving this request, the RS takes the following steps.

6.1.1 DETERMINE PERMISSIONS

The RS examines the request and determines what Permissions (resource/scopes) are required to fulfill this request. This step requires that the RS determine all the resources that would be in the result of the request, loop through all of them, and map them to UMA permissions.

6.1.2 REGISTER PERMISSIONS WITH THE PRINCIPAL AS

The RS contacts the Principal AS at its Permission Registration Endpoint and registers the determined Permissions. If this is successful, the Principal AS returns a string, which we call the Principal AS Ticket, in the response.

6.1.3 REFERRAL RESPONSE TO THE CLIENT

The RS sends the client an Unauthorized (HTTP 401) response with the respective headers pointing to the Principal AS from which the client must obtain the Principal AS Access Token. This response also includes the Ticket.

6.2 CLIENT'S FIRST REQUEST FOR ACCESS TOKEN FROM THE PRINCIPAL AS

The client, then, requests an Access Token from the Principal AS at its Authorization Endpoint. This request includes the Principal AS Ticket received in the previous step from the RS and a set of Claims Tokens (signed by trusted issuers). Note that the UMA protocol allows for an array of such token to be provided, so it is possible to include several Claims Tokens, for example, one vouching for the identity of the Client itself, and another one vouching for the identity of the end-user behind the Client.

After receiving this request, the Principal AS, takes the following steps.

6.2.1 VERIFICATION

The Principal AS verifies the presented Principal AS Ticket to ensure it is valid and corresponds to an earlier permission registration.

It also verifies the presented Claims Tokens to ensure they are valid and signed by a trusted issuer.

Finally, it examines its policies to determine whether the requested access is authorized. The result could be:

- A deny, in which case the Principal AS sends a Forbidden response (HTTP 403) to the client.
- A permit, in which case the Principal AS issues an Access Token corresponding to the registered Permissions and sends it back to the client.
- An obligation to require the approval of the Secondary AS which leads to a referral to the Secondary AS.

6.2.2 REGISTER PERMISSIONS WITH SECONDARY AS

In the latter case, when the approval of the Secondary AS is required per policies, the Principal AS contacts the Secondary AS at its Permission Registration Endpoint and registers the Permissions requested by the Client. In response, the Secondary AS sends back a Secondary AS Ticket as a receipt.

Note that, in general, the Principal AS has the power to modify the registered permission based on its polices and configuration. For example, it may register a subset of the initial requested permissions based on some policies, or translate them to the vocabulary understood by the Secondary AS based on the interoperability configurations.

6.2.3 REFERRAL RESPONSE TO THE CLIENT

The Principal AS sends an Unauthorized response (HTTP 401) to the client, including a pointer to the Secondary AS and the Secondary AS Ticket.

6.3 CLIENT’S REQUEST TO GET A SECONDARY AS ACCESS TOKEN

The client, then, sends a request for a Secondary AS Access Token to the Secondary AS at its Authorization Endpoint. This request includes the Secondary AS Ticket and a set of Claims Token signed by a trusted issuers. Note that the UMA protocol allows for an array of such token to be provided, so it is possible to include several Claims Tokens, for example, one vouching for the identity of the Client itself, and another one vouching for the identity of the end-user behind the Client.

The Secondary AS takes the following steps to process this request.

6.3.1 VERIFICATION

The Secondary AS verifies the presented Secondary AS Ticket to ensure it is valid and corresponds to an earlier permission registration.

It also verifies the Claims Tokens to ensure they are valid and signed by trusted issuers.

Finally, it searches to find the policies pertaining to the requested Permissions and examines them to determine whether this access is authorized. If the result is a deny, the Secondary AS sends a Forbidden response (HTTP 403) to the Client. Otherwise, it will issue a Secondary AS Access Token, as discussed below.

6.3.2 ISSUING A SECONDARY AS ACCESS TOKEN AND RESPONSE TO THE CLIENT

In the case of a Permit response, the Secondary AS determines the granted Scopes to the Client and issues an Access Token associated with those Scopes. Note that, based on the Secondary AS policies, this can result in granting all or a subset of the Scopes initially requested in the Permission Registration step.

6.4 CLIENT'S SECOND REQUEST FOR ACCESS TOKEN FROM THE PRINCIPAL AUTHORIZATION SERVER

After receiving the Secondary AS Access Token, the Client requests an Access Token from the Principal AS again, this time, including the Consent Access Token, denoting that access was approved by the Consent AS. Note that the Consent Access Token in this case must be included in the body of the HTTP request, since the Authorization Header already bears a different bearer token for authorizing access to the endpoint. The most suitable place to include the Secondary AS Access Token is in the Claims Token array since the approval of the Secondary AS can be considered as a trusted claim.

The Principal AS, subsequently, takes the following steps to process this request.

6.4.1 VERIFICATION

The Principal AS verifies the presented Principal AS Ticket again to ensure it is valid and corresponds to an earlier permission registration.

It also verifies the other Claims Tokens again to ensure they are valid and signed by a trusted issuer.

The Principal AS also invokes the Token Introspection flow with the Secondary AS to ensure that the presented Secondary AS Access Token is valid, and also to receive the granted Permissions associated with it.

6.4.2 SCOPE RECONCILIATION, ISSUING AN ACCESS TOKEN AND RESPONSE TO THE CLIENT

If all of the above steps are successful, the Principal AS will determine the final granted Scopes, issue an Access Token, and send it back to the Client.

A crucial step here is a Scope Reconciliation step, in which the Principal AS examine the granted scopes by the Secondary AS and verifies them with its own policies and granted scopes. Thus, the Principal AS has the power to deny some of those granted scopes, per overarching policies, or translate them from the vocabulary of the Secondary AS, to a different vocabulary understood by the RS, per configurations.

6.5 CLIENT'S SECOND REQUEST TO RESOURCE SERVER WITH ACCESS TOKEN

After receiving the Access Token from the Principal AS, the Client repeats its request to the RS, this time presenting the Access Token. The RS, then, runs a Token Introspection flow with the Principal AS to ensure that the Access Token is valid, and to determine the granted Permissions/Scopes associated with it. Once these Permissions/Scopes are determined, the RS invokes the Security Labeling Service and Privacy Preserving Service on the response to ensure that only the information matching the granted scopes are included. Finally, the RS sends the labeled (and potentially modified) copy of the data to the Client.

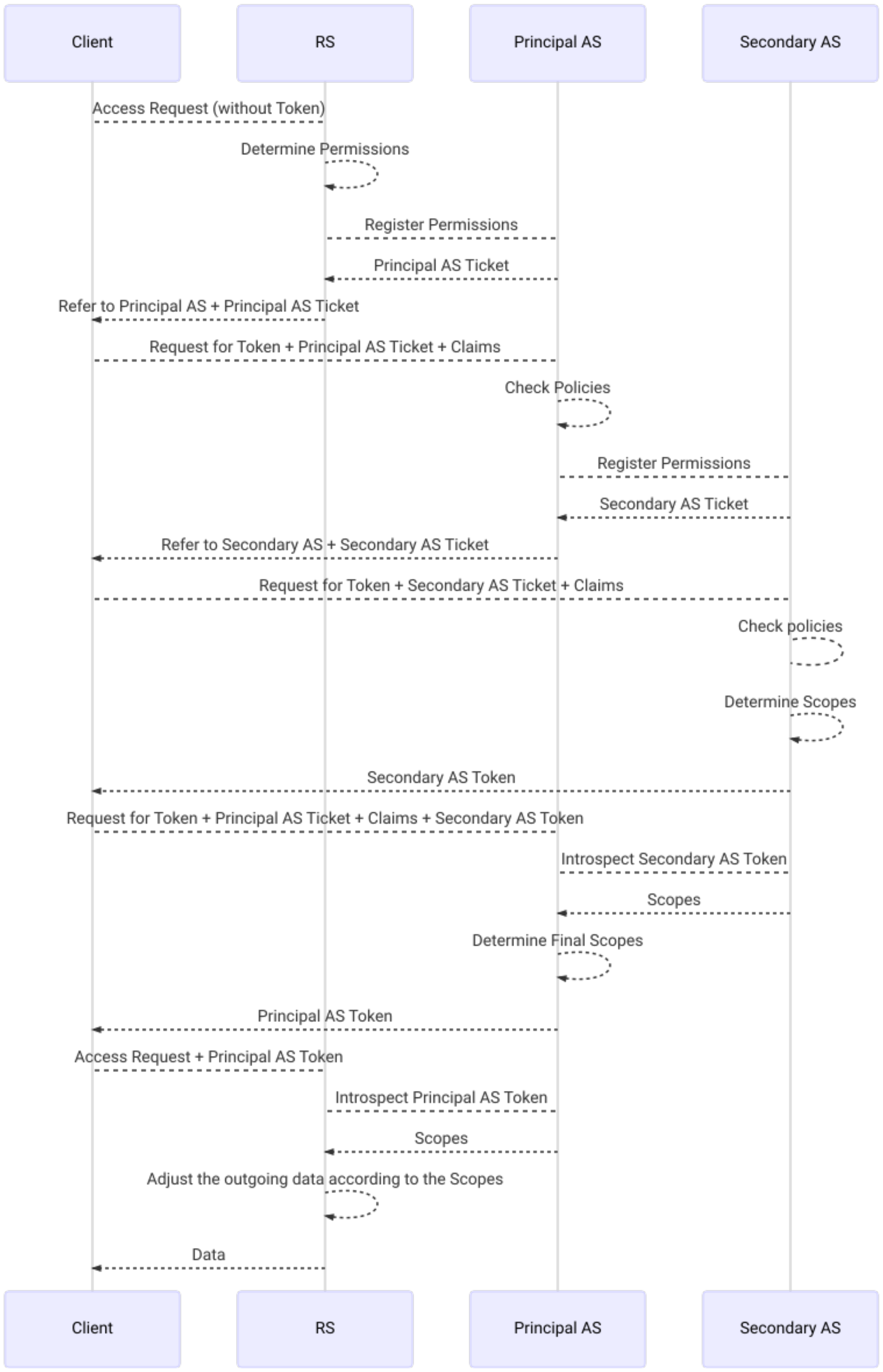
Client
RS
Principal AS
Secondary AS
Access Request (without Token)
Determine Permissions
Register Permissions
Principal AS Ticket
Refer to Principal AS + Principal AS Ticket
Request for Token + Principal AS Ticket + Claims
Check Policies
Register Permissions
Secondary AS Ticket
Refer to Secondary AS + Secondary AS Ticket
Request for Token + Secondary AS Ticket + Claims
Check policies
Determine Scopes
Secondary AS Token
Request for Token + Principal AS Ticket + Claims + Secondary AS Token
Introspect Secondary AS Token
Scopes
Determine Final Scopes
Principal AS Token
Access Request + Principal AS Token
Introspect Principal AS Token
Scopes
Adjust the outgoing data according to the Scopes
Data
Client
RS
Principal AS
Secondary AS

Figure 2. Detailed sequence diagram of a successful cascaded authorization flow.

7 UMA 2.0 AND CASCADED AUTHORIZATION

Since the results of this project was presented, there has been an update planned for the UMA standard. In this section, we discuss the major changes to the standard and how they affect the Cascaded Authorization proposal.

7.1 UMA 2.0

In this section, we briefly discuss the changes in the upcoming UMA 2.0.

7.1.1 PROTECTION API

The proposed UMA 2.0 moves the original Protection API out of the main standard thereby leaving the communication between the Resource Server and the UMA servers up to the discretion of the respective developers. However, there is a new profile called Federated UMA, which describes and standardizes the former Protection API. Thus, developers who would like to follow a standard for the communication between the RA and AS, can follow the Federated UMA profile.

One of the major improvements to the Protection API in this profile is that unlike the older version of UMA, at the time of the Permission Registration, the RS now has the option to register more than one Permissions. This is important because depending on the Client's request, the set of resources requested by the client may belong to more than one registered resource set, especially when the Client sends a request in the form of a query asking for several resources.

7.1.2 FEDERATED UMA AND MULTIPLE AUTHORIZATION SERVERS

The new Federated UMA profile states that the RS has the option to protect the resources in its custody using different Authorization Servers. In other words, the Resource Owner, or the RS itself, can designate one AS (e.g. AS_1) to protecting resources of type A, while designating a different AS (e.g. AS_2) for protecting resource of type B. Therefore, when receiving a request by the Client that calls for sending resources of type A, the RS will refer the Client to AS_1 and when the request involves resource of type B, the RS will refer the Client to AS_2. The current UMA 2.0 draft does not specify the details of such flow and deems it out of the scope of the standard.

7.2 UMA 2.0 AND CASCADED AUTHORIZATION

While the Federated UMA profile touches on the subject of multiple Authorization Servers, it does not discuss the details of the flow for such a use-case, and as mentioned above, considers it out of the scope of the profile.

Moreover, while the profile recognizes the use-case in which different Authorization Servers must be consulted for authorizing access to different groups of resources, it does not consider to the case where multiple Authorization Servers must be consulted in responding to a *single* request and it still implies that in processing each request, the RS will end up selecting one single AS and proceed the UMA flow with that AS.

This is an important use-case and the assumption that at the end of the day, each request will have to be processed with a single AS seems to be restrictive and failing to cover some use-cases such as FHIR resources.

For example, consider the case above where the Resource Owner has designated AS_1 for resources of type A (e.g. R_A) while resources of type B (e.g. R_B) are assigned to another AS, AS_2.

Now, if a Client sends a request in the form of a query, the response to which includes a bundle of resources of both type A and B (e.g. both R_A and R_B), then the RS has to refer the Client to acquire RPTs from both RS_1 and RS_2. While, the current Federated UMA profile does not cover how the flow will look like in this case, the Cascaded Authorization proposal describes this in detail.

Also note that in the current Federated UMA profile, all the following logics are left to the RS:

1. Determining what Authorization Server(s) must be consulted in response to a request,
2. Following the Permission Registration flow with each of them,
3. Following the Token Introspection flow with each of them, and
4. Combining the decisions and granted scopes from all of the consulted Authorization Servers and consolidating them into an ultimate decision.

This means that the RS, in such cases, will have to have a rather bloated authorization module which is involved in making some policy decisions (Step 1 and 4). This is somehow contrary to the goal of offloading the authorization policy processing from the RS which seems to be one of the major goals of UMA.

The Cascaded Authorization proposal, on the other hand, factors out this logic and moves it onto the Principal AS, therefore, leaving the RS almost entirely clear of authorization decisions.

8 USE-CASE EXAMPLE: PATIENT CONSENT

This section discusses an example of using Cascaded Authorization in the use-case of releasing information authorized by a patient under the jurisdictional policies of Patient Right of Access. The main scenario of this use-case is that the patient grants a client (or a group thereof), such as a Mobile Application, the right to access their health information residing at the servers of one or more custodian organizations. Any client covered by this grant, then, should be able to get a copy of the information belonging to that patient under the constraints set by the patient.

In this use-case we will assume that the Principal AS will enforce the overarching policies, including the jurisdictional policy pertaining to Patient Right of Access. Patient Consents Directives, including the request for granting access under Right of Access provisions, will be filed and processed by a separate AS, which we call the Consent AS.

An overview of this flow is provided in Figure 3. Figure 4 shows the sequence diagram in line with the general flow of the Cascaded Authorization.

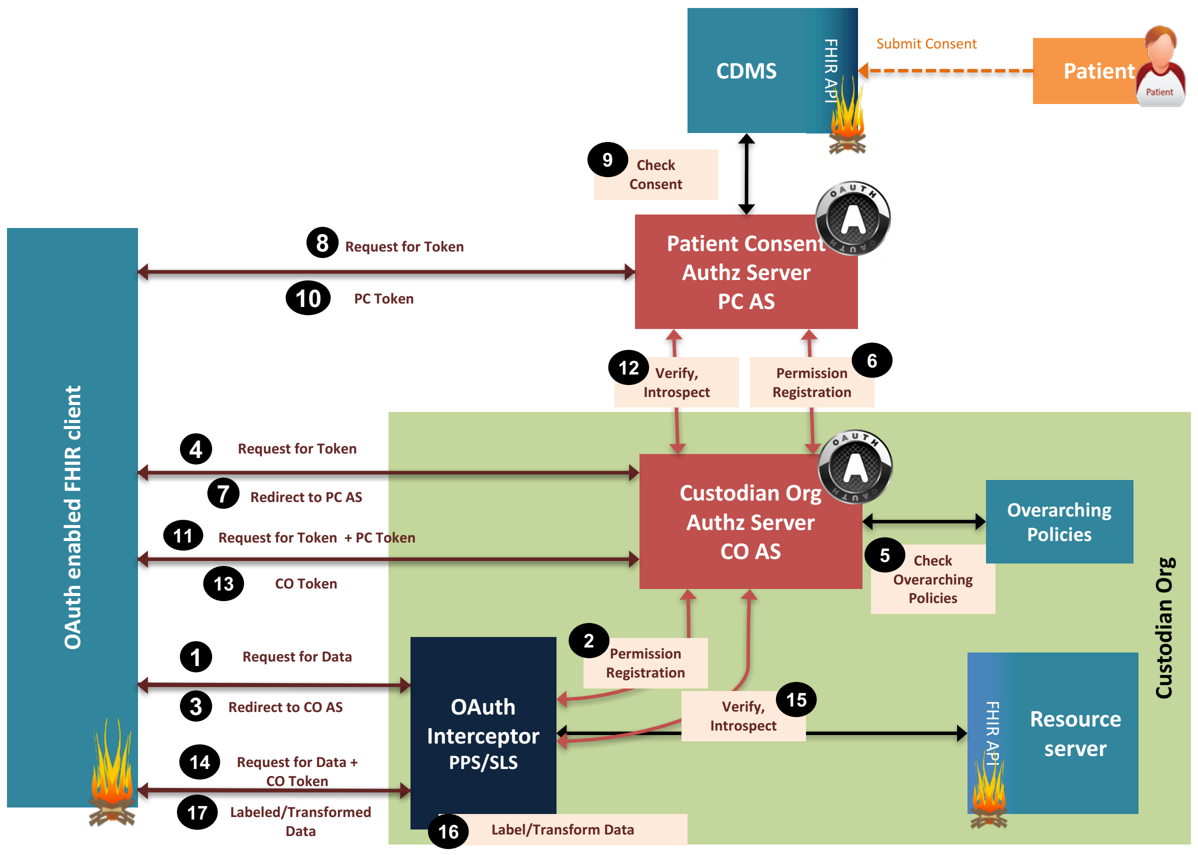

Figure 3. Overview of the Flow for Patient Consent Using Cascaded Authorization

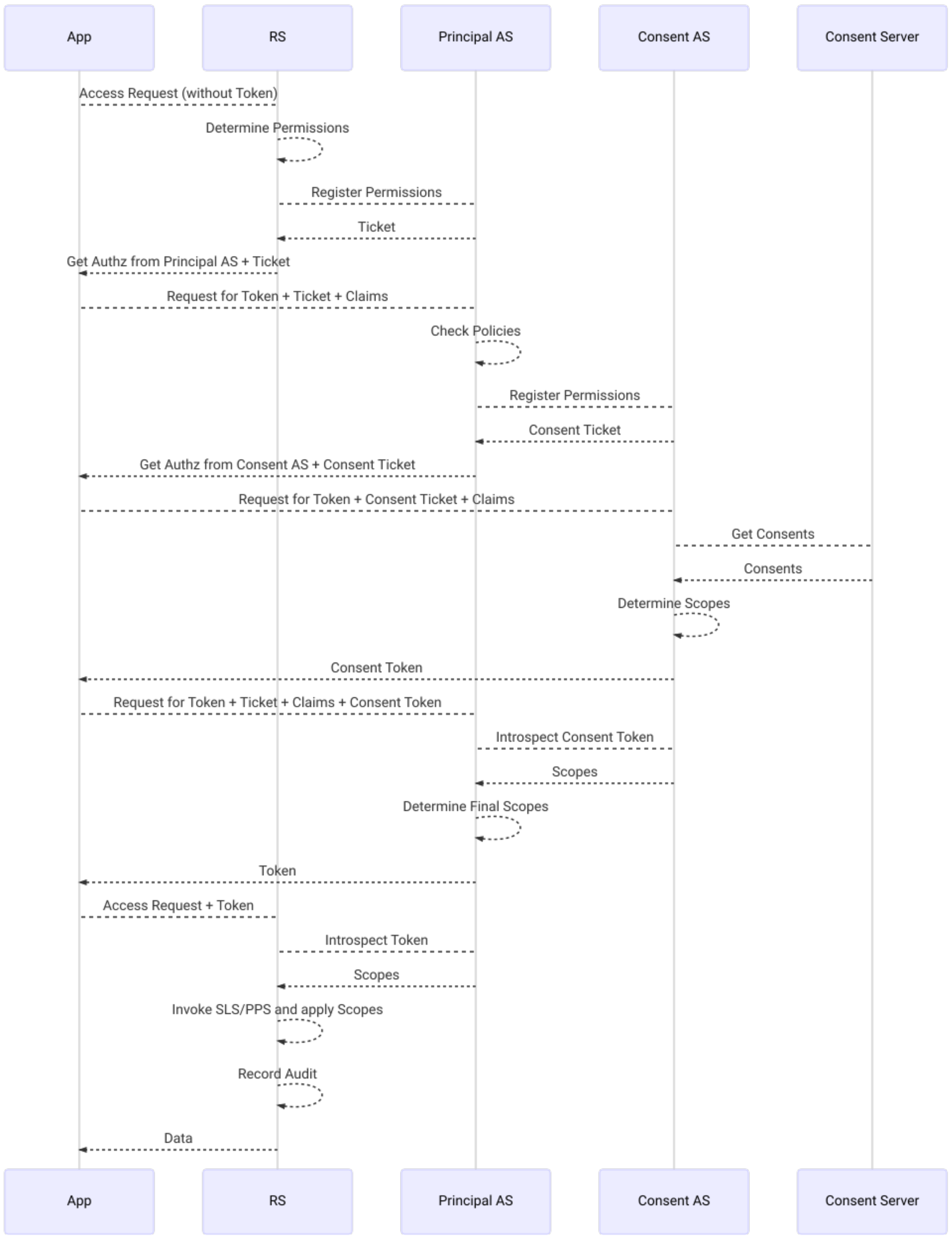

Figure 4. Sequence Diagram for the Patient Consent Use-Case.