Correlated Authorization—Draft

Igor Zboran
izboran@gmail.com

***Abstract*—*Correlated authorization* is used to build trust between security domains during the conveyance of user information from the identity provider to the authorization server.**

This paper introduces *correlated authorization* as a dual-authority mail trust framework built on top of User-Managed Access (UMA) [1, 2] and OAuth 2.0 [3] protocols that allows users (resource owners) to delegate access to other users (requesting parties) across security domain boundaries. The requesting party is responsible for creating the request, while the resource owner approves this request either when it is online or by creating a policy. The resource owner and the requesting party may belong to different security domains administered by the respective authorities.

The proposed concept uses a permission ticket issued by the resource owner's authorization server as a correlation handle that binds the requesting party's claims to the authorization process. An email address is used as the unique requesting party identifier. The requesting party authenticates to the resource owner's authorization server using a challenge-response authentication protocol, while the push-pull mechanism elevates trust between the respective authorities. On the requesting party side, *correlated authorization* uses the token exchange extension of OAuth 2.0 protocol [4] as a counterpart to the UMA protocol.

Given these capabilities, *correlated authorization* is becoming an essential enabler of email infrastructure modernization.

I. Introduction

With the growing popularity of protocols based on the OAuth 2.0 [3] specification, there is a need for an interoperable standard that specifies how to convey information about the user from an identity provider to an authorization server, especially across security domain boundaries. The problem is that such a system is difficult to design because OAuth 2.0 [3], OIDC [5] and UMA are single-authority protocols. This draft profiles and combines the OAuth 2.0 and UMA protocols into a dual-authority framework, which not only meets the needs of interoperability, but also elevates trust between mutually unknown parties.

II. Motivation

Correlated authorization is an attempt to revive UMA WG's original idea—UMA wide ecosystem [6], when the resource owner and requesting party might "know each other" in the real world, but the resource owner's authorization server has no pre-established trust with the requesting party or any of their identity/claims providers—in other words, when the resource owner's authorization server and requesting party's identity provider don't know each other.

III. UMA Wide Ecosystem Concept

This high-level view illustrated in Figure 1 gives you an idea of relationships between UMA wide ecosystem entities.

UMA uses a special jargon. For the sake of brevity of this paper, the following list of acronyms will be used:

- IdP - Identity Provider
- AS - Authorization Server
- RS - Resource Server
- RO - Resource Owner
- RqP - Requesting Party
- RPT - Requesting Party Token

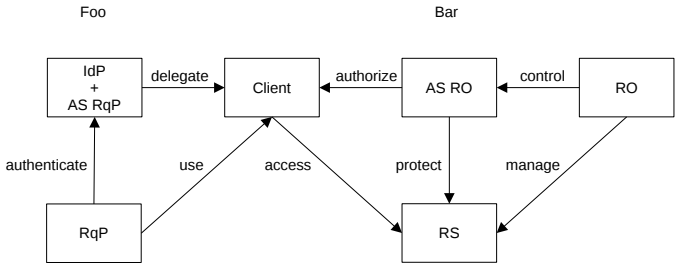


Fig. 1. Relationships between UMA wide ecosystem entities

The UMA wide ecosystem concept uses relationship-driven policies to drive automated dual-authority authorization assessment and token issuance. The relationship-driven policies incorporate user-to-user (U2U) relationships and user-to-resource (U2R) relationships.

IV. Challenge-Response Authentication Protocol

Figure 2 shows the unilateral entity authentication protocol [7] adapted for the *correlated authorization* framework that links an authenticator with the verifier through the client, and allows the claimant to convey identity information to the verifier.

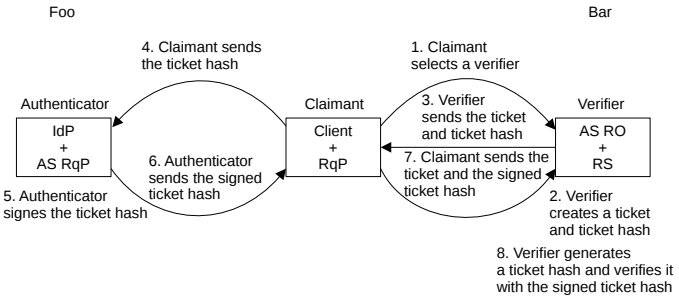


Fig. 2. Unilateral entity authentication protocol

Successful completion of steps means that the claimant has authenticated itself to the verifier. The ticket represents the random challenge, and the signed ticket hash represents the response. Ticket hash is used here to ensure that the actual value of the ticket is not disclosed to the authenticator.

V. Sequence Diagram

The following sequence diagram describes the mechanism and policies of the *correlated authorization* framework, which utilizes the UMA protocol with the token exchange extension of OAuth 2.0 [4], where an

access token is used to obtain a claims token from the Security Token Service (STS) endpoint.

UMA Profile

The sequence diagram[1] illustrated in Figure 3 represents a profile of the UMA protocol and is in full compliance with the UMA 2.0 specification[2].

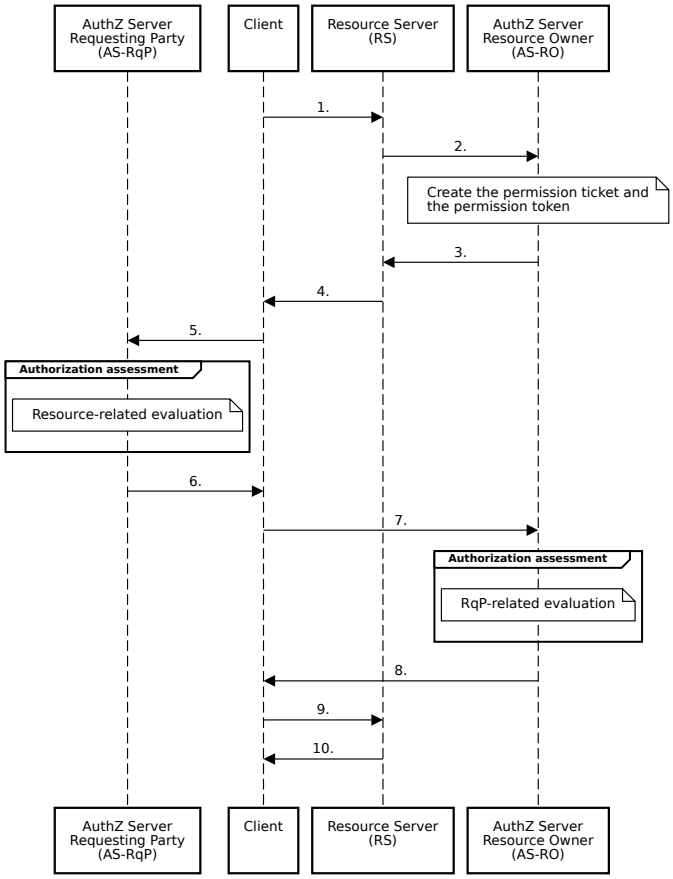


Fig. 3. *Correlated authorization* sequence diagram

Prerequisites:

- The AS-RqP supports the OAuth 2.0 Token Exchange [4] extension of OAuth 2.0.
- The AS-RqP publishes its metadata on a URL /.well-known/oauth-authorization-server (alternatively on /.well-known/openid-configuration).
- The AS-RqP also acts as RqP's Identity Provider.
- The client is registered at the AS-RqP as a public or confidential client and acts as a Relying Party in a RqP's Identity Provider in order to obtain an access token with user claims.
- The client should be registered at the AS-RO as a public or confidential client; in case of immediate access, the client does not have to be registered at the AS-RO.
- The RO has set up the RS and registers his resource at the AS-RO to get his resource_uri according to the UMA Federated Authorization [2] specification.

Steps:

1. The RqP directs the client to access the resource_uri with no access token.
2. The RS requests a permission ticket.

   The AS generates the permission ticket itself (ticket is a random NONCE) and the permission token[3], which is bound to the permission ticket through a permission ticket hash. The permission token contains these claims: {issuer, ts, rs_uri, resource_uri_hash, permission_ticket_hash} where
   - issuer is the URI that identifies who issues the permission token
   - ts is the timestamp of when the permission ticket was created
   - audience is the URI that identifies the resource server
   - resource_uri_hash = Base64URL-Encode(SHA256(resource_uri))
   - permission_ticket_hash = Base64URL-Encode(SHA256(permission_ticket))

3. The AS returns the permission ticket and the permission token.
4. Without an access token, the RS will return HTTP code 401 (Unauthorized) with the permission ticket and the permission token.
5. The client requests a claims token by presenting the access token with user claims, permission token and resource uri (token exchange request).

   {grant_type = token-exchange,
   resource = resource_uri,
   scope = permission_token
   subject_token = access_token_with_user_claims,
   subject_token_type = urn:ietf:params:oauth:token-type:access_token,
   requested_token_type = urn:ietf:params:oauth:token-type:jwt}

   The AS-RqP performs an authorization assessment
   - 1. verify permission_token signature
   - 2. extract resource_uri_hash claim from permission_token
   - 3. compare resource_uri_hash vs. Base64URL-Encode(SHA256(resource_uri))
   - 4. evaluate issuer, ts, audience, resource_uri

   The AS-RqP generates the claim token, which contains these claims: {user_claims, permission_ticket_hash} where
   - user_claims are extracted from access_token_with_user_claims
   - permission_ticket_hash is extracted from permission_token

6. After an authorization assessment, it is positive, the AS-RqP returns the claims token.
7. At the AS-RO the client requests an RPT by presenting the claims token and the permission ticket.

   {grant_type = uma-ticket,
   pushed_claims = claims_token}

   The AS-RO performs an authorization assessment

1. A more detailed diagram is shown on the last page.
2. Unlike the UMA specification, the *correlated authorization* framework allows the use of the UMA grant with or without client authentication or identification. Whether or not to allow unauthenticated or unidentified clients are policy decisions that are at the discretion of the authorization server.
3. The permission token is not mentioned in the UMA specification. A detailed description of the permission token format is out of scope of this paper.

1. verify permission_ticket
2. extract user_claims from claims_token
3. select email_address claim
4. bootstrap discovery of AS-RqP config url from email address via WebFinger; if this doesn't work, build well-known url using domain part of email_address
5. verify claims_token signature
6. evaluate resource_uri
7. extract permission_ticket_hash claim from claims_token
8. compare permission_ticket_hash vs. Base64URL-Encode(SHA256(permission_ticket))
9. evaluate user_claims

8. After an authorization assessment, it is positive, the AS-RO returns RPT.
9. With the valid RPT the client tries to access the resource_uri.
10. The RS validates the RPT, it is valid, the RS allow access the protected resource.

VI. Push-Pull Trust Elevation

It is recommended to use a push-pull mechanism to increase trust. It means that the resource owner first sends a link[4] to their shared resources to the requesting party. To do this, the requesting party must have its resource server registered at its authorization server, and needs to have its resource server accessible in the form of a well-known resource_uri e.g., mailto:john.doe@example.com for anyone. Here, the requesting party also acts as the resource owner of his resource server. After receiving the resource link, the requesting party's authorization server must set the policy correctly, either by the requesting party itself or automatically by the agent. Only then can the requesting party download the resources from the resource owner's resource server. Such a push-pull mechanism elevates trust between the resource owner's authoritative domain and requesting party's authoritative domain.

Generally speaking, the push-pull trust elevation mechanism can utilize existing email infrastructure, or a new secure web-based communication infrastructure needs to be built based on *correlated authorization* itself. The new-built infrastructure should mirror existing email infrastructure.

VII. Authority Boundaries, Interactions and Scenarios

The *correlated authorization* framework allows us to indirectly (through the client) link identity providers with authorization servers governed by different authorities that are not required to share information or collaborate. The following scenarios demonstrate a system of trust between multiple authorities that allows the conveyance of identity information from identity providers to authorization servers across security domain boundaries.

A. Identity Federation Scenario

The scenario illustrated in Figure 4 allows you to link a single authorization server to multiple identity providers. The client falls under the governance of the resource owner's respective authority.

4. In general, the link to shared resources may be transfered to the requesting party through any trusted channel, e.g., using store-and-forward systems such as the mail system that uses the push—fire-and-forget Simple Mail Transfer Protocol (SMTP). In any case, the shared resource uri should contain a random string.

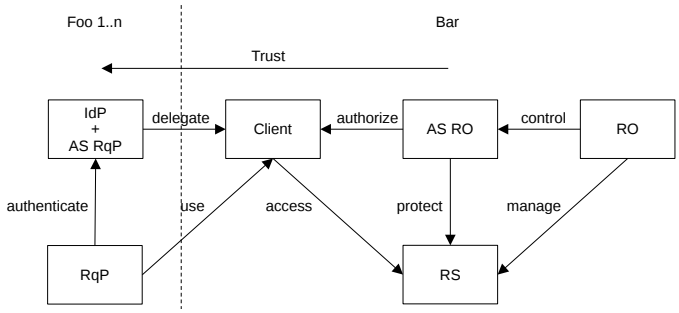


Fig. 4. Identity federation scenario

The identity federation with many-to-one topology uses third-party identity providers. The requesting party can operate across resource servers govern by single resource owner's respective authority. The push-pull trust elevation mechanism is not applicable in this scenario.

B. Data Federation Scenario

The data federation scenario illustrated in Figure 5 allows you to link a single identity provider to multiple authorization servers. The client falls under the governance of the requesting party's respective authority.

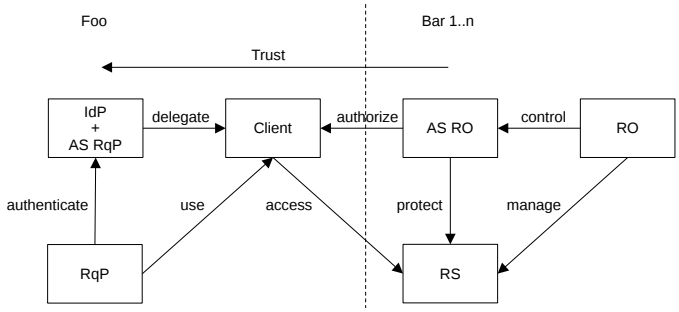


Fig. 5. Data federation scenario

The data federation with one-to-many topology uses third-party authorization servers. The requesting party can operate across many resource servers, each of which is governed by a different respective authority of resource owners.

C. Mesh Federation Scenario

As the name suggests, the scenario illustrated in Figure 6 allows multiple authorization servers to be linked to multiple identity providers. The client does not fall under the governance of the resource owner's respective authority nor the requesting party's respective authority.

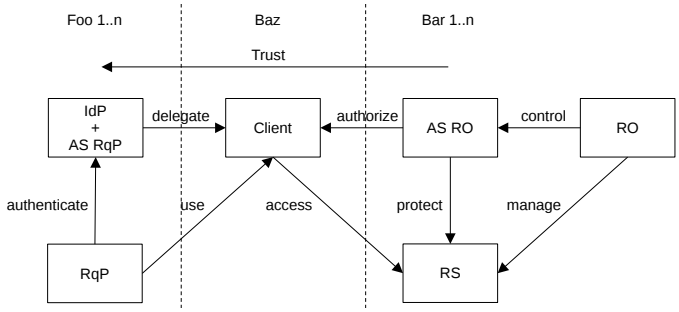


Fig. 6. Mesh federation scenario

The mesh federation with many-to-many topology uses third-party identity providers and third-party authorization servers. The requesting party can operate across many resource servers govern by many resource owners' respective authorities.

VIII. Use Cases

Secure cross-domain data exchange systems. In particular, Authorization-Enhanced Mail System [8]. Furthermore, file sharing, instant messaging, teleconferencing. Also, Healthcare systems, Fintech and Telco services.

IX. Conclusion and Future Work

The UMA philosophy of the resource owner and the requesting party projected onto the *correlated authorization* trust framework, matches the philosophy of the sender and recipient of the mail system. In fact, the *correlated authorization* concept has been designed with the Authorization-Enhanced Mail System [8] in mind. The following are potential future R&D areas:

1. Define relationship-driven policies—user-to-user and user-to-resource relationships.
2. Provide more details on the push-pull mechanism.

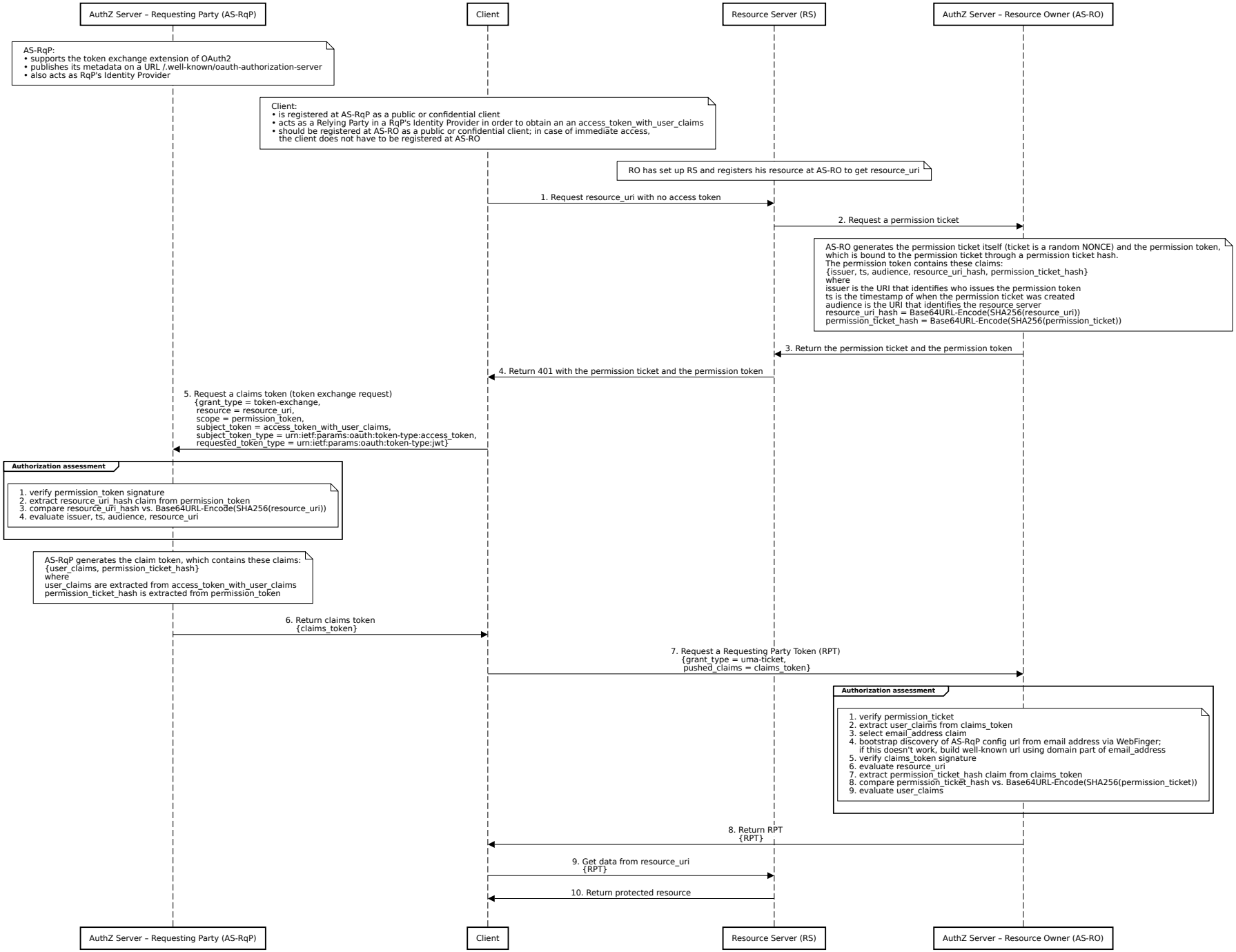
A prototype implementation of the proposed framework, working as a proof of concept, would be interesting to build.

Acknowledgment

This work has benefited from the valuable discussions with Eve Maler, founder of WG-UMA [9]; and Alec Laws, chair of WG-UMA [9]. Both gave feedback that improved this paper's content. Last but not least, the UMA Work Group archives [10, 11] serve as a source of comprehensive information on authorization-related topics—many thanks to all involved.

References

[1] E. Maler, M. Machulak, J. Richer, and T. Hardjono, "User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization," Internet Engineering Task Force (2019), https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html.
[2] E. Maler, M. Machulak, J. Richer, and T. Hardjono, "Federated Authorization for User-Managed Access (UMA) 2.0" Internet Engineering Task Force (2019), https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-federated-authz-2.0.html.
[3] E. D. Hardt, "The OAuth 2.0 Authorization Framework," IETF RFC 6749 (Informational), 2012, http://tools.ietf.org/html/rfc6749.
[4] M. Jones, A. Nadalin, B. Campbell, J. Bradley, C. Mortimore, "OAuth 2.0 Token Exchange," RFC 8693 (2020), https://rfc-editor.org/rfc/rfc8693.txt.
[5] OpenID specifications at "OpenID Foundation," 2022, https://openid.net/developers/specs/.
[6] "UMA telecon 2016-03-31" https://kantarainitiative.org/confluence/display/uma/UMA+telecon+2016-03-31
[7] National Institute of Standards and Technology, "FIPS PUB 196: Entity Authentication Using Public Key Cryptography," 1997. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/196/archive/1997-02-18/documents/fips196.pdf.
[8] I. Zboran "Authorization-Enhanced Mail System" GitHub repository https://github.com/umalabs/authorization-enhanced-mail-system.
[9] "User-Managed Access" Work Group at "Kantara Initiative" https://kantarainitiative.org/confluence/display/uma/Home.
[10] "The WG-UMA Archives" https://kantarainitiative.org/pipermail/wg-uma/.
[11] "Kantara Initiative User Managed Access WG" https://groups.google.com/g/kantara-initiative-uma-wg.


AuthZ Server – Requesting Party (AS-RqP)
Client
Resource Server (RS)
AuthZ Server – Resource Owner (AS-RO)
AS-RqP:
• supports the token exchange extension of OAuth2
• publishes its metadata on a URL /.well-known/oauth-authorization-server
• also acts as RqP's Identity Provider
Client:
• is registered at AS-RqP as a public or confidential client
• acts as a Relying Party in a RqP's Identity Provider in order to obtain an an access_token_with_user_claims
• should be registered at AS-RO as a public or confidential client; in case of immediate access,
the client does not have to be registered at AS-RO
RO has set up RS and registers his resource at AS-RO to get resource_uri
1. Request resource_uri with no access token
2. Request a permission ticket
AS-RO generates the permission ticket itself (ticket is a random NONCE) and the permission token,
which is bound to the permission ticket through a permission ticket hash.
The permission token contains these claims:
{issuer, ts, audience, resource_uri_hash, permission_ticket_hash}
where
issuer is the URI that identifies who issues the permission token
ts is the timestamp of when the permission ticket was created
audience is the URI that identifies the resource server
resource_uri_hash = Base64URL-Encode(SHA256(resource_uri))
permission_ticket_hash = Base64URL-Encode(SHA256(permission_ticket))
3. Return the permission ticket and the permission token
4. Return 401 with the permission ticket and the permission token
5. Request a claims token (token exchange request)
{grant_type = token-exchange,
resource = resource_uri,
scope = permission_token,
subject_token = access_token_with_user_claims,
subject_token_type = urn:ietf:params:oauth:token-type:access_token,
requested_token_type = urn:ietf:params:oauth:token-type:jwt}
Authorization assessment
1. verify permission_token signature
2. extract resource_uri_hash claim from permission_token
3. compare resource_uri_hash vs. Base64URL-Encode(SHA256(resource_uri))
4. evaluate issuer, ts, audience, resource_uri
AS-RqP generates the claim token, which contains these claims:
{user_claims, permission_ticket_hash}
where
user_claims are extracted from access_token_with_user_claims
permission_ticket_hash is extracted from permission_token
6. Return claims token
{claims_token}
7. Request a Requesting Party Token (RPT)
{grant_type = uma-ticket,
pushed_claims = claims_token}
Authorization assessment
1. verify permission_ticket
2. extract user_claims from claims_token
3. select email_address claim
4. bootstrap discovery of AS-RqP config url from email address via WebFinger;
if this doesn't work, build well-known url using domain part of email_address
5. verify claims_token signature
6. evaluate resource_uri
7. extract permission_ticket_hash claim from claims_token
8. compare permission_ticket_hash vs. Base64URL-Encode(SHA256(permission_ticket))
9. evaluate user_claims
8. Return RPT
{RPT}
9. Get data from resource_uri
{RPT}
10. Return protected resource
AuthZ Server – Requesting Party (AS-RqP)
Client
Resource Server (RS)
AuthZ Server – Resource Owner (AS-RO)