Correlated Authorization—Draft

Igor Zboran
izboran@gmail.com

***Abstract*—Trustworthiness of data—determined by its provenance—is fundamental to the cybersecurity of the Internet as a whole. As business boundaries are redrawn, it is becoming increasingly clear that a flexible, trust framework is needed to support the digital economy.**

This paper introduces Correlated Authorization as a dual-authority trust framework built on top of User-Managed Access (UMA) [1, 2] and OAuth 2.0 [3, 4] protocols that allow users (resource owners) to delegate access to other users (requesting parties) across the 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 holder-of-key assertion mechanism establishes trust between the respective authorities. On the requesting party side, Correlated Authorization uses the token exchange extension of the OAuth 2.0 protocol [4] as a counterpart to the UMA protocol.

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 establishes 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 the 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 do not 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 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
- PAT - Protection API Access Token
- RPT - Requesting Party Token

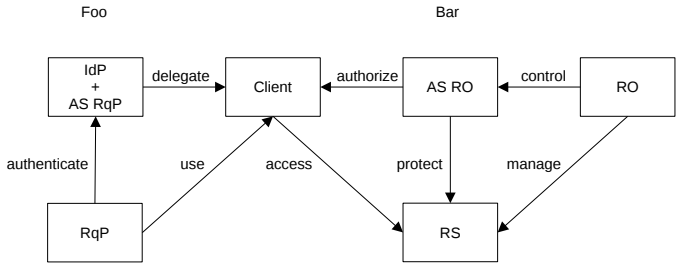

Fig. 1. Relationships between UMA wide ecosystem entities

The UMA wide ecosystem concept uses relationship-driven policies to drive automated dual-authority trust 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.

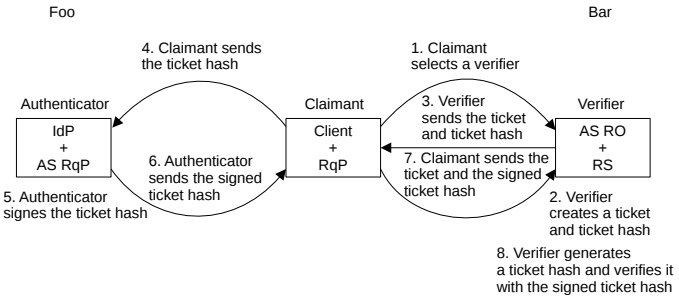

Fig. 2. Unilateral entity authentication protocol

- The authenticator is represented by the requesting party's identity provider and the requesting party's authorization server.
- The verifier is represented by the resource owner's authorization server and the resource owner's resource server.
- The claimant represents a principal which communicates with both the authenticator and the verifier using a client.

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

V. Federation and Assertions

The Correlated Authorization framework uses a multiparty federation protocol, as illustrated in Figure 3. There are some similarities and a few differences between the Correlated Authorization federation scenario and the federation scenario described in NIST SP 800-63C [8] document.

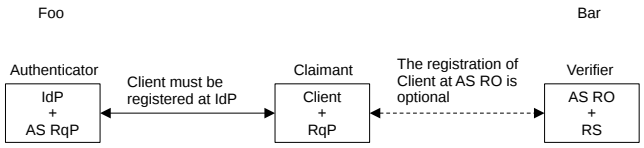

Fig. 3. Multiparty federation protocol

A. Federation Model

Two two-party relationships are formed in a Correlated Authorization federation protocol, as shown in Figure 3. The first one is between the claimant and the authenticator, pre-established, e.g., by a dynamic registration. The second one is between the claimant and the verifier, and can be ephemeral, established dynamically. The claimant communicates with both the authenticator and the verifier using a client. The verifier and the authenticator communicate with each other through the client, which acts as a transparent bridge that links the authenticator to the verifier.

Authenticator may establish whitelists of verifiers authorized to receive authentication, and verifiers may establish whitelists of authenticators that the verifier will accept authentication. Authenticators may also establish blacklists of verifiers not authorized to receive authentication, and verifiers may also establish blacklists of authenticators that the verifier will not accept authentication.

B. Assertions

According the UMA Grant specification [1], Section 3.3.1, the potential types of claim token formats are ID Tokens and SAML assertion. The Correlated Authorization framework uses the urn:ietf:params:oauth:client-assertion-type:jwt-bearer format for the asserted pushed claims tokens. To prove that the claimant is the rightful subject of the assertion the holder-of-key assertion [9] is used, which contains a hash value of the ticket possessed by the claimant.

According the RFC7521 [9] "Assertion authorization grants may be used with or without client authentication or identification. Whether or not client authentication is needed in conjunction with an assertion authorization grant, as well as the supported types of client authentication, are policy decisions at the discretion of the authorization server.". It follows that the UMA grant with assertions may be used without client authentication. This only applies for specific scenarios. Be careful.

VI. 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 an identity claims token from the Security Token Service (STS) endpoint.

A. UMA Profile

The sequence diagram (see Appendix A for a detailed diagram) illustrated in Figure 4 represents a profile of the UMA protocol and is in full compliance with the UMA 2.0 specification. 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.

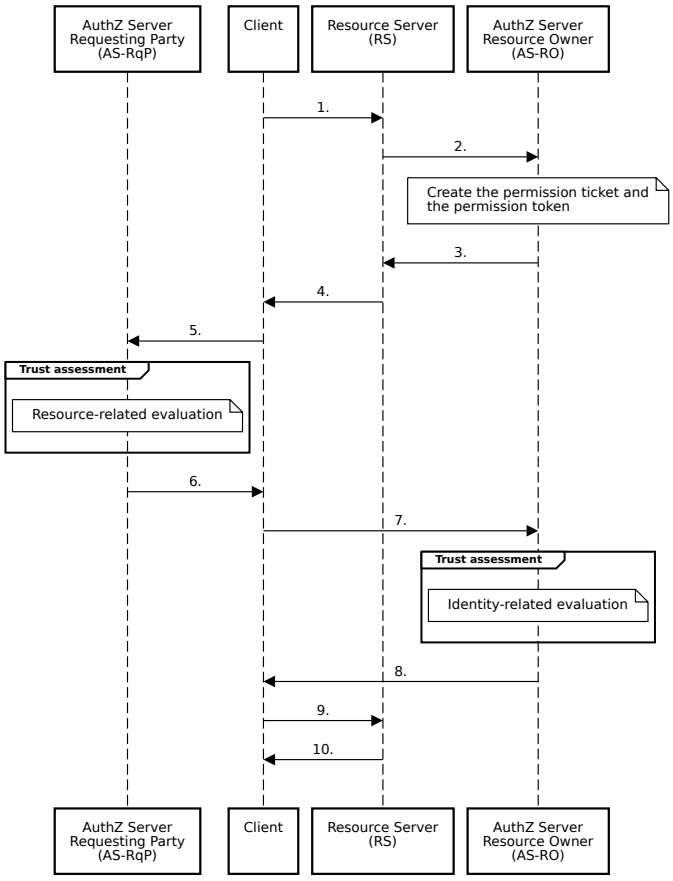

Fig. 4. Correlated Authorization sequence diagram — UMA profile

Prerequisites:

- The AS-RqP supports the OAuth 2.0 Token Exchange [4] extension of OAuth 2.0, as an STS service.
- 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 an RqP's Identity Provider in order to obtain an access token with user claims.
- The client can be registered at the AS-RO as a public or confidential client. The registration is optional.
- 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 and the Resource Description extension.
- The RO sets policies to the resource sets with the authorization server to indicate who can access the resources.

Steps:

1. The RqP directs the client to access the resource_uri, e.g. to get or post data, with no access token.
2. Using a valid PAT the RS requests a permission ticket and resource claims token.

   The AS generates the permission ticket itself (ticket is a random NONCE) and resource claims token, which is bound to the permission ticket through a permission ticket hash. The resource claims token contains these claims:

   {issuer, ts, audience, email_address, action, resource_uri_hash, permission_ticket_hash}

   where

   - issuer is the URI that identifies who issues the resource claims token
   - ts is the timestamp of when the permission ticket was created
   - audience is the URI that identifies the resource server
   - email_address is the email address of the resource owner
   - action is the HTTP request method
   - resource_uri_hash = Base64URL-Encode(SHA256(resource_uri))
   - permission_ticket_hash = Base64URL-Encode(SHA256(permission_ticket))

   The resource claims token is not mentioned in the UMA specification. A detailed description of the resource claims token format is out of the scope of this paper.

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

   {grant_type = token-exchange,
   resource = resource_uri,
   scope = resource_claims_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 a trust assessment by evaluating the resource URI provenance/ownership
   1. select the email_address claim from resource_claims_token
   2. bootstrap discovery of AS-RO url from email address via WebFinger; if this does not work, build well-known url using domain part of email_address
   3. compare AS-RO url with the iss claim from resource_claims_token
   4. verify the resource_claims_token signature
   5. extract resource_uri_hash claim from resource_claims_token
   6. compare resource_uri_hash vs. Base64URL-Encode(SHA256(resource_uri))
   7. evaluate the remaining resource claims

   The AS-RqP generates the identity claim token, which contains these claims:

   {audience, user_claims, permission_ticket_hash}

   where

   - audience = the iss claim value extracted from resource_claims_token
   - user_claims are extracted from access_token_with_user_claims
   - permission_ticket_hash is extracted from resource_claims_token

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

   {grant_type = uma-ticket,
   ticket = ticket, claim_token = identity_claims_token}

   The AS-RO performs a trust assessment by evaluating the RqP identity provenance/ownership
   1. verify the permission_ticket
   2. verify the audience claim from identity_claims_token
   3. extract user_claims from identity_claims_token
   4. select the email_address claim from user_claims
   5. bootstrap discovery of AS-RqP url from email address via WebFinger; if this does not work, build well-known url using domain part of email_address
   6. compare AS-RO url with the iss claim from resource_claims_token
   7. verify the identity_claims_token signature
   8. extract permission_ticket_hash claim from identity_claims_token
   9. compare permission_ticket_hash vs. Base64URL-Encode(SHA256(permission_ticket))
   10. evaluate the remaining identity claims

8. After a trust assessment, it is positive, the AS-RO returns RPT and optionally a refresh token.
9. With the valid RPT the client tries to access the resource_uri to get or post data.
10. The RS validates the RPT; it is valid, the RS allows access to the protected resource.

VII. Coarse-Grained Authorization

The centralized AS-RqP policy verifies the provenance of the resource claims in order to issue the identity claims token. The centralized AS-RO policy verifies the provenance of the identity claims in order to issue the requesting party token. The authorization decisions are made at the resource owner's authorization server and the requesting party's authorization server using an Attribute-Based Access Control (ABAC) system, which is shown in Figure 5.

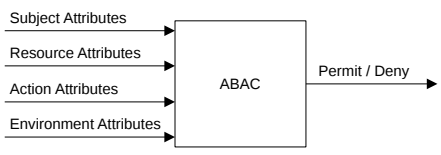

Fig. 5. Attribute-Based Access Control system

The input of the ABAC system is grouped into four categories:

- Subject attributes about the user making request, taken from the access token or identity claims token.
- Resource attributes about the resource being accessed, taken from resource claims token or defined by the ticket.
- Action attributes about the HTTP request method (GET, POST, PUT, DELETE).
- Attributes about the context in which the operation is taking place, e.g., level of assurance.

The output of the ABAC system is an allow or deny decision.

VIII. Fine-Grained Access Control

To avoid overloading the authorization server, information about which user has access to which resources, together with a set of permissions that define what each user can do, should be stored on the resource server in the access control list (ACL). ACLs should be managed through the resource server API. It is also possible to use a local ABAC engine on the resource server, where the Policy Decision Point (PDP) can go through the ACL and the Policy Information Points (PIP) can use groups or roles resolved from the RPT. Given that, the final authorization decision is made on the resource server.

IX. 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 the security domain boundaries.

A. Identity Federation Scenario

The scenario illustrated in Figure 6 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.

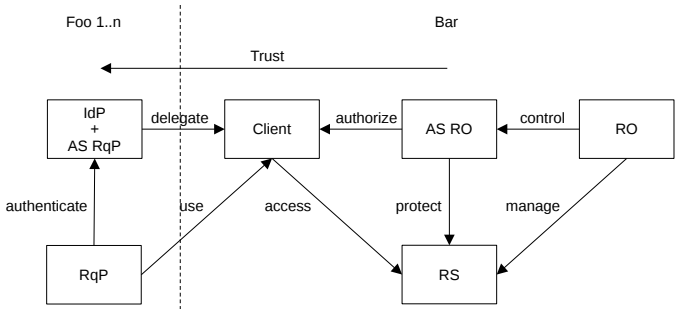

Fig. 6. Identity federation scenario

The identity federation with many-to-one topology uses third-party identity providers. The requesting party can operate across resource servers governed by a single resource owner's respective authority.

B. Data Federation Scenario

The data federation scenario illustrated in Figure 7 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.

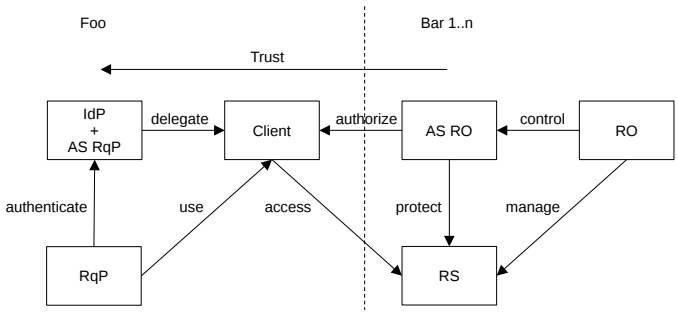

Fig. 7. Data federation scenario

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 8 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.

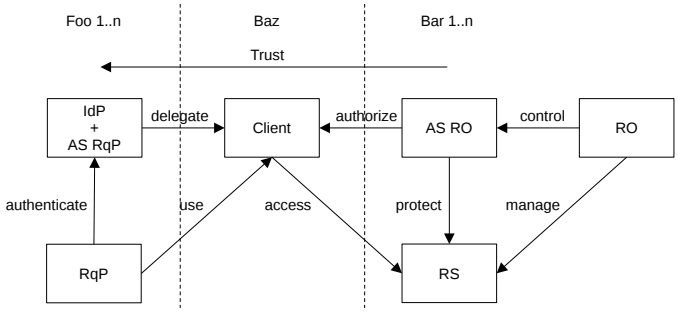

Fig. 8. Mesh federation scenario

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 governed by many resource owners' respective authorities.

X. Applications and Use Cases

The Correlated Authorization framework may be used to secure cross-domain data exchange systems. In particular, Authorization-Enhanced Mail System [10]. Furthermore, file sharing, instant messaging, teleconferencing. Also, Healthcare systems, Fintech, and Telco services.

XI. 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 [10] in mind. The following are potential future R&D areas:

1. Move the Correlated Authorization flow to the RFC.
2. Explore other ways of data origin authenticity (WebFinger, DKIM). Use the DKIM signed email in a claims token as an expedient way to convey some asserted attributes about the sender (alternatively, after forwarding, about the recipient also).

The ability to dynamically establish ephemeral trust between the requesting party and the resource owner's resources makes the Correlated Authorization framework compatible with the Zero Trust concept.

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 [11], and Alec Laws, chair of WG-UMA [11]. Both gave feedback that improved this paper's content. Last but not least, the UMA Work Group archives [12, 13] 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] "Digital Identity Guidelines: Federation and Assertions", NIST Special Publication 800-63C, June 2017, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63c.pdf.

[9] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, http://www.rfc-editor.org/info/rfc7521..

[10] I. Zboran "Authorization-Enhanced Mail System," GitHub repository, March 2022, https://github.com/umalabs/authorization-enhanced-mail-system/releases/download/v0.1/Authorization-Enhanced_Mail_System.pdf.

[11] "User-Managed Access" Work Group at "Kantara Initiative," https://kantarainitiative.org/confluence/display/uma/Home.

[12] "The WG-UMA Archives," https://kantarainitiative.org/pipermail/wg-uma/.

[13] "Kantara Initiative User-Managed Access WG," https://groups.google.com/g/kantara-initiative-uma-wg.

APPENDIX A.


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, as an STS service.
• 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
• is registered at AS-RO as a public or confidential client
RO has set up RS and registers his resource at AS-RO to get resource_uri and sets policies
to the resource sets with the authorization server to indicate who can access the resources.
1. Send HTTP request to resource_uri with no access token
2. Request a permission ticket and resource claims token
{PAT}
AS-RO generates the permission ticket itself (ticket is a random NONCE) and the resource claims token,
which is bound to the permission ticket through a permission ticket hash.
The resource claims token contains these claims:
{issuer, ts, audience, email_address, resource_uri_hash, permission_ticket_hash}
issuer is the URI that identifies who issues the resource claims token
ts is the timestamp of when the permission ticket was created
audience is the URI that identifies the resource server
email_address is the email address of the resource owner
action is the HTTP request method
resource_uri_hash = Base64URL-Encode(SHA256(resource_uri))
permission_ticket_hash = Base64URL-Encode(SHA256(permission_ticket))
3. Return the permission ticket and resource claims token
4. Return 401 with the permission ticket and resource claims token
5. Request an identity claims token (token exchange request)
{grant_type = token-exchange,
resource = resource_uri,
scope = resource_claims_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}
Trust assessment – evaluate the resource URI provenance / ownership
1. select the email_address claim from resource_claims_token
2. bootstrap discovery of AS-RO url from email address via WebFinger;
if this doesn't work, build well-known url using domain part of email_address
3. compare AS-RO url with the iss claim from resource_claims_token
4. verify the resource_claims_token signature
5. extract resource_uri_hash claim from resource_claims_token
6. compare resource_uri_hash vs. Base64URL-EnCode(SHA256(resource_uri))
7. evaluate the remaining resource claims
AS-RqP generates the identity claims token, which contains these claims:
{audience, user_claims, permission_ticket_hash}
audience = the iss claim value extracted from resource_claims_token
user_claims are extracted from access_token_with_user_claims
permission_ticket_hash is extracted from resource_claims_token
6. Return the identity claims token
{identity_claims_token}
7. Request a Requesting Party Token (RPT)
{grant_type = uma-ticket,
ticket = ticket,
claim_token = identity_claims_token}
Trust assessment – evaluate the RqP identity provenance / ownership
1. verify the permission_ticket
2. verify the audience claim from identity_claims_token
3. extract user_claims from identity_claims_token
4. select the email_address claim from user_claims
5. bootstrap discovery of AS-RqP url from email address via WebFinger;
if this doesn't work, build well-known url using domain part of email_address
6. compare AS-RqP url with the iss claim from resource_claims_token
7. verify the identity_claims_token signature
8. extract permission_ticket_hash claim from identity_claims_token
9. compare permission_ticket_hash vs. Base64URL-Encode(SHA256(permission_ticket))
10. evaluate the remaining identity claims
8. Return RPT, optionally an refresh token
{RPT}
9. Send HTTP request to resource_uri
{RPT}
10. Return HTTP response typically with 200, 201 status code
AuthZ Server – Requesting Party (AS-RqP)
Client
Resource Server (RS)
AuthZ Server – Resource Owner (AS-RO)