Workgroup: GNAP
Internet-Draft:
draft-richer-transactional-authz-10
Published: 2 September 2020
Intended Status: Standards Track
Expires: 6 March 2021
Authors: J. Richer, Ed.
Bespoke Engineering

XYZ: Grant Negotiation Access Protocol

Abstract

This document defines a mechanism for delegating authorization to a piece of software, and conveying that delegation to the software. This delegation can include access to a set of APIs as well as information passed directly to the software.

This document is input into the GNAP working group and should be referred to as "XYZ" to differentiate it from other proposals.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 6 March 2021.

Table of Contents

1. Protocol

This protocol allows a piece of software to request delegated authorization to an API, protected by an authorization server usually on behalf of a resource owner. The user operating the software may interact with the authorization server to authenticate, provide consent, and authorize the request.

The process by which the delegation happens is known as a grant, and the GNAP protocol allows for the negotiation of the grant process over time by multiple parties

1.1. Roles

The Authorization Server (AS) manages the requested delegations for the RO. The AS issues tokens and directly delegated information to the RC. The AS is defined by its grant endpoint, a single URL that accepts a POST request with a JSON payload. The AS could also have other endpoints, including interaction endpoints and user code endpoints, and these are introduced to the RC as needed during the delegation process.

The Resource Client (RC, aka "client") requests tokens from the AS and uses tokens at the RS. The RC is identified by its key, and can be known to the AS prior to the first request. The AS determines which policies apply to a given client.

The Resource Server (RS) accepts tokens from the RC and validates them (potentially at the AS). The RS serves delegated resources on behalf of the RO.

The Resource Owner (RO) authorizes the request from the RC to the RS, often interactively at the AS.

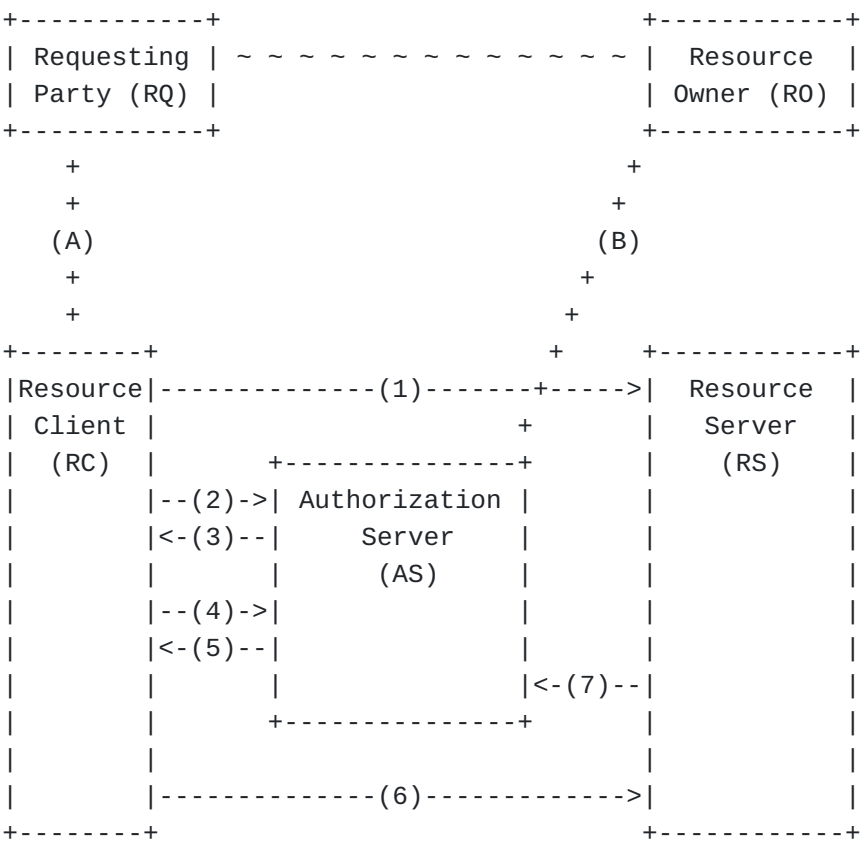
The Requesting Party (RQ, aka "user") operates the RC and may be the same party as the RO in many circumstances.

1.2. Sequences

The GNAP protocol can be used in a variety of ways to allow the core delegation process to take place. Many portions of this process are conditionally present depending on the context of the deployments, and not every step in this overview will happen in all circumstances.

Note that a connection between roles in this process does not necessarily indicate that a specific protocol message is sent across the wire between the components fulfilling the roles in question, or

that a particular step is required every time. In some circumstances, the information needed at a given stage is communicated out-of-band or is pre-configured between the components or entities performing the roles. For example, one entity can fulfil multiple roles, and so explicit communication between the roles is not necessary within the protocol flow.



Legend
+ + + indicates a possible interaction with a human
----- indicates an interaction between protocol roles
~ ~ ~ indicates a potential equivalence or communication between rol

*(A) The RQ interacts with the RC to indicate a need for resources on behalf of the RO. This could identify the RS the RC needs to call, the resources needed, or the RO that is needed to approve the request. Note that the RO and RQ are often the same entity in practice.

*(1) The RC attempts to call the RS (Section 10.4) to determine what access is needed. The RS informs the RC that access can be granted through the AS.

*(2) The RC creates requests access at the AS (Section 2).

*(3) The AS processes the request and determines what is needed to fulfill the request. The AS sends its response to the RC (Section 3).

*(B) If interaction is required, the AS interacts with the RO (Section 4) to gather authorization. The interactive component of the AS can function using a variety of possible mechanisms including web page redirects, applications, challenge/response protocols, or other methods. The RO approves the request for the RC being operated by the RQ. Note that the RO and RQ are often the same entity in practice.

*(4) The RC continues the grant at the AS (Section 5).

*(5) If the AS determines that access can be granted, it returns a response to the RC (Section 3) including an access token (Section 3.2) for calling the RS and any directly returned information (Section 3.4) about the RO.

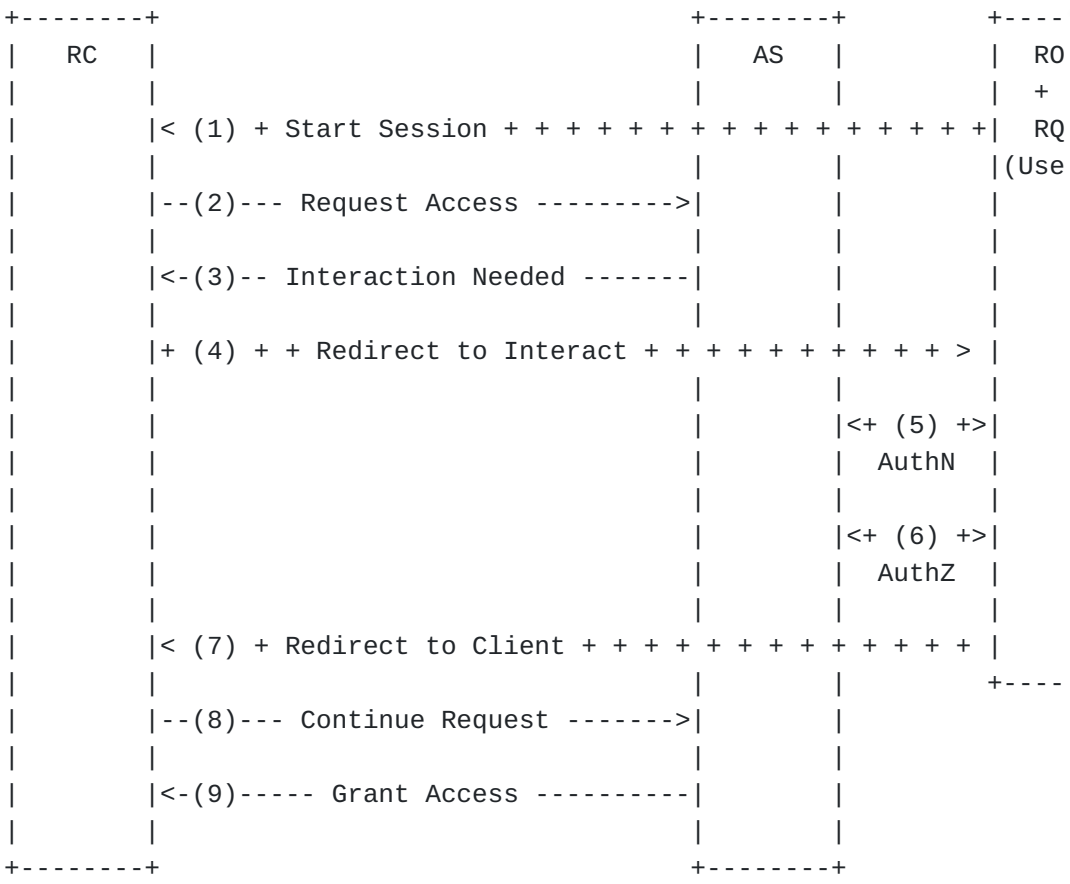
*(6) The RC uses the access token (Section 7) to call the RS.

*(7) The RS determines if the token is sufficient for the request by examining the token, potentially calling the AS (Section 10.1).

The following sections and Appendix C contain specific guidance on how to use the GNAP protocol in different situations and deployments.

1.2.1. Redirect-based Interaction

In this example flow, the RC is a web application that wants access to resources on behalf of the current user, who acts as both the requesting party (RQ) and the resource owner (RO). Since the RC is capable of directing the user to an arbitrary URL and receiving responses from the user's browser, interaction here is handled through front-channel redirects using the user's browser. The RC uses a persistent session with the user to ensure the same user that is starting the interaction is the user that returns from the interaction.



1. The RC establishes a verifiable session to the user, in the role of the RQ.

2. The RC requests access to the resource (Section 2). The RC indicates that it can redirect to an arbitrary URL (Section 2.5.1) and receive a callback from the browser (Section 2.5.3). The RC stores verification information for its callback in the session created in (1).

3. The AS determines that interaction is needed and responds (Section 3) with a URL to send the user to (Section 3.3.1) and information needed to verify the callback (Section 3.3.3) in (7). The AS also includes information the RC will need to continue the request (Section 3.1) in (8). The AS associates this continuation information with an ongoing request that will be referenced in (4), (6), and (8).

4. The RC stores the verification and continuation information from (3) in the session from (1). The RC then redirects the user to the URL (Section 4.1) given by the AS in (3). The user's browser loads the interaction redirect URL. The AS loads the pending request based on the incoming URL generated in (3).

5. The user authenticates at the AS, taking on the role of the RO.

6. As the RO, the user authorizes the pending request from the RC.

7. When the AS is done interacting with the user, the AS redirects the user back (Section 4.4.1) to the RC using the callback URL provided in (2). The callback URL is augmented with an interaction reference that the AS associates with the ongoing request created in (2) and referenced in (4). The callback URL is also augmented with a hash of the security information provided in (2) and (3). The RC loads the verification information from (2) and (3) from the session created in (1). The RC calculates a hash (Section 4.4.3) based on this information and continues only if the hash validates.

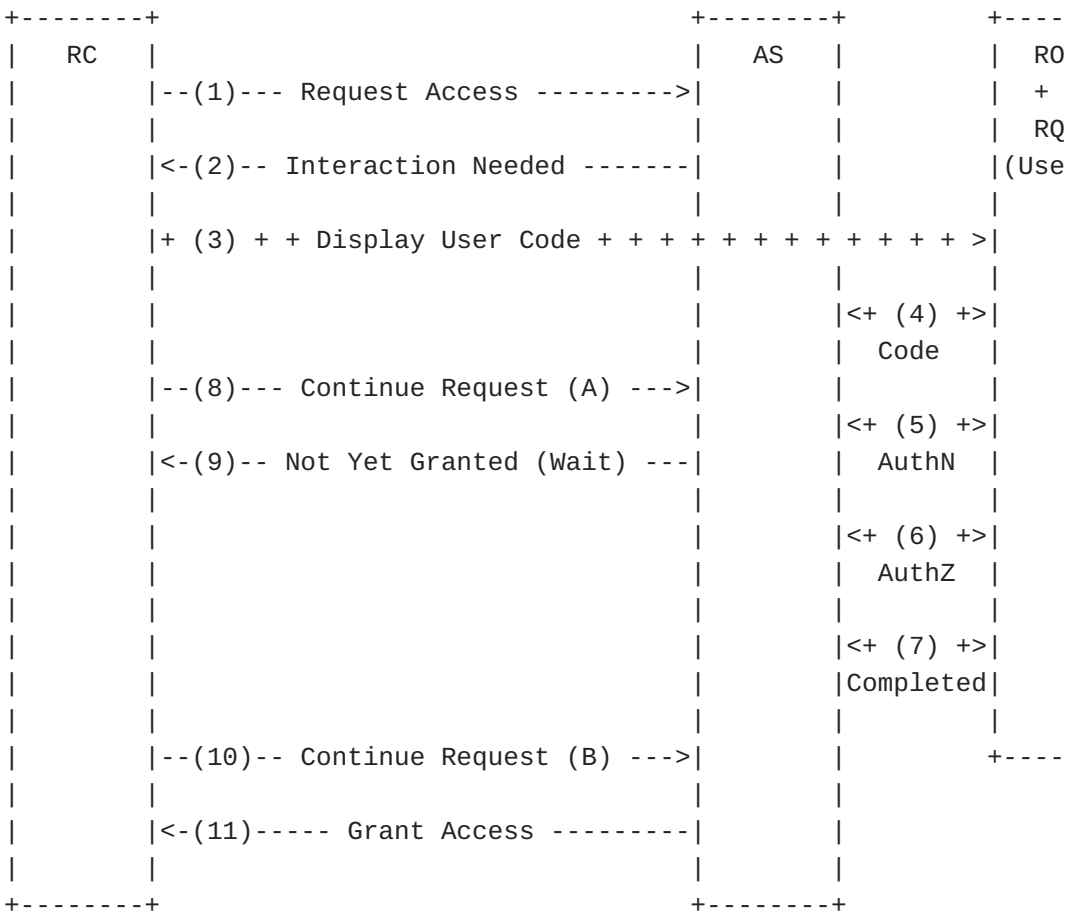
8. The RC loads the continuation information from (3) and sends the interaction reference from (7) in a request to continue the request (Section 5.1). The AS validates the interaction reference ensuring that the reference is associated with the request being continued.

9. If the request has been authorized, the AS grants access to the information in the form of access tokens (Section 3.2) and direct subject information (Section 3.4) to the RC.

An example set of protocol messages for this method can be found in Appendix C.1.

1.2.2. User-code-based Interaction

In this example flow, the RC is a device that is capable of presenting a short, human-readable code to the user and directing the user to enter that code at a known URL. The RC is not capable of presenting an arbitrary URL to the user, nor is it capable of accepting incoming HTTP requests from the user's browser. The RC polls the AS while it is waiting for the RO to authorize the request. The user's interaction is assumed to occur on a secondary device. In this example it is assumed that the user is both the RQ and RO, though the user is not assumed to be interacting with the RC through the same web browser used for interaction at the AS.



1. The RC requests access to the resource (Section 2). The RC indicates that it can display a user code (Section 2.5.4).

2. The AS determines that interaction is needed and responds (Section 3) with a user code to communicate to the user (Section 3.3.4). This could optionally include a URL to direct the user to, but this URL should be static and so could be configured in the RC's documentation. The AS also includes information the RC will need to continue the request (Section 3.1) in (8) and (10). The AS associates this continuation information with an ongoing request that will be referenced in (4), (6), (8), and (10).

3. The RC stores the continuation information from (2) for use in (8) and (10). The RC then communicates the code to the user (Section 4.1) given by the AS in (2).

4. The user's directs their browser to the user code URL. This URL is stable and can be communicated via the client's documentation, the AS documentation, or the client software itself. The client does not provide a mechanism to launch the user's browser at this URL. The user enters the code

communicated in (3) to the AS. The AS validates this code against a current request in process.

5. The user authenticates at the AS, taking on the role of the RO.

6. As the RO, the user authorizes the pending request from the RC.

7. When the AS is done interacting with the user, the AS indicates to the user that the request has been completed.

8. Meanwhile, the RC loads the continuation information stored at (3) and continues the request (Section 5). The AS determines which ongoing access request is referenced here and checks its state.

9. If the access request has not yet been authorized by the RO in (6), the AS responds to the RC to continue the request (Section 3.1) at a future time through additional polling. This response can include refreshed credentials as well as information regarding how long the RC should wait before calling again. The RC replaces its stored continuation information from the previous response (2).

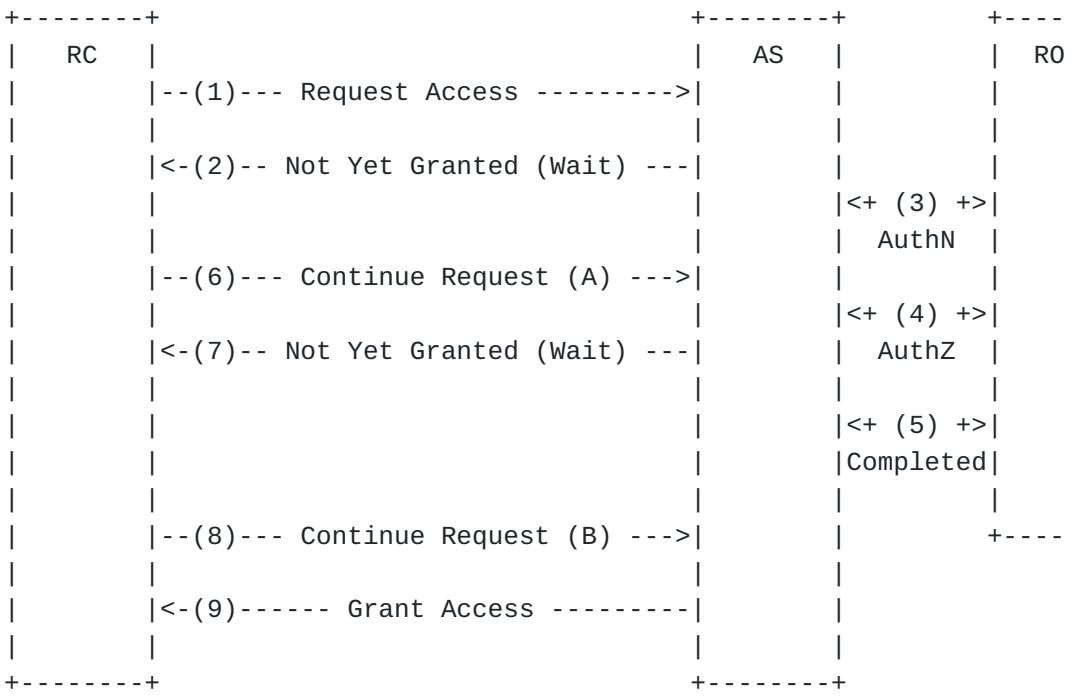
10. The RC continues to poll the AS (Section 5) with the new continuation information in (9).

11. If the request has been authorized, the AS grants access to the information in the form of access tokens (Section 3.2) and direct subject information (Section 3.4) to the RC.

An example set of protocol messages for this method can be found in Appendix C.2.

1.2.3. Asynchronous Authorization

In this example flow, the RQ and RO roles are fulfilled by different parties, and the RO does not interact with the RC. The AS reaches out asynchronously to the RO during the request process to gather the RO's authorization for the RC's request. The RC polls the AS while it is waiting for the RO to authorize the request.



1. The RC requests access to the resource (Section 2). The RC does not send any interactions capabilities to the server, indicating that it does not expect to interact with the RO. The RC can also signal which RO it requires authorization from, if known, by using the user request section (Section 2.4).

2. The AS determines that interaction is needed, but the RC cannot interact with the RO. The AS responds (Section 3) with the information the RC will need to continue the request (Section 3.1) in (6) and (8), including a signal that the RC should wait before checking the status of the request again. The AS associates this continuation information with an ongoing request that will be referenced in (3), (4), (5), (6), and (8).

3. The AS determines which RO to contact based on the request in (1), through a combination of the user request (Section 2.4), the resources request (Section 2.1), and other policy information. The AS contacts the RO and authenticates them.

4. The RO authorizes the pending request from the RC.

5. When the AS is done interacting with the user, the AS indicates to the user that the request has been completed.

6. Meanwhile, the RC loads the continuation information stored at (3) and continues the request (Section 5). The AS determines which ongoing access request is referenced here and checks its state.

7. If the access request has not yet been authorized by the RO in (6), the AS responds to the RC to continue the request (Section 3.1) at a future time through additional polling. This response can include refreshed credentials as well as information regarding how long the RC should wait before calling again. The RC replaces its stored continuation information from the previous response (2).

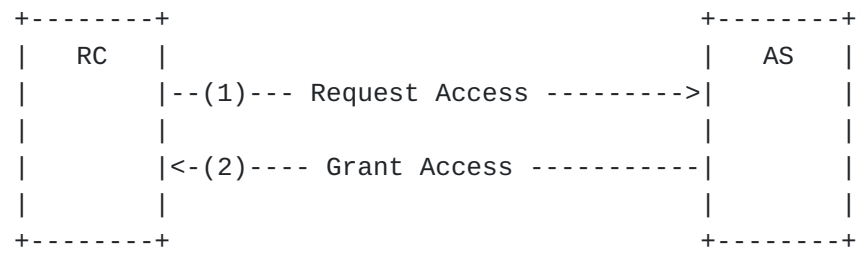
8. The RC continues to poll the AS (Section 5) with the new continuation information in (7).

9. If the request has been authorized, the AS grants access to the information in the form of access tokens (Section 3.2) and direct subject information (Section 3.4) to the RC.

An example set of protocol messages for this method can be found in Appendix D.1.

1.2.4. Software-only Authorization

In this example flow, the AS policy allows the RC to make a call on its own behalf, without the need for a RO to be involved at runtime to approve the decision. The Since there is no explicit RO, the RC does not interact with an RO.



1. The RC requests access to the resource (Section 2). The RC does not send any interactions capabilities to the server.

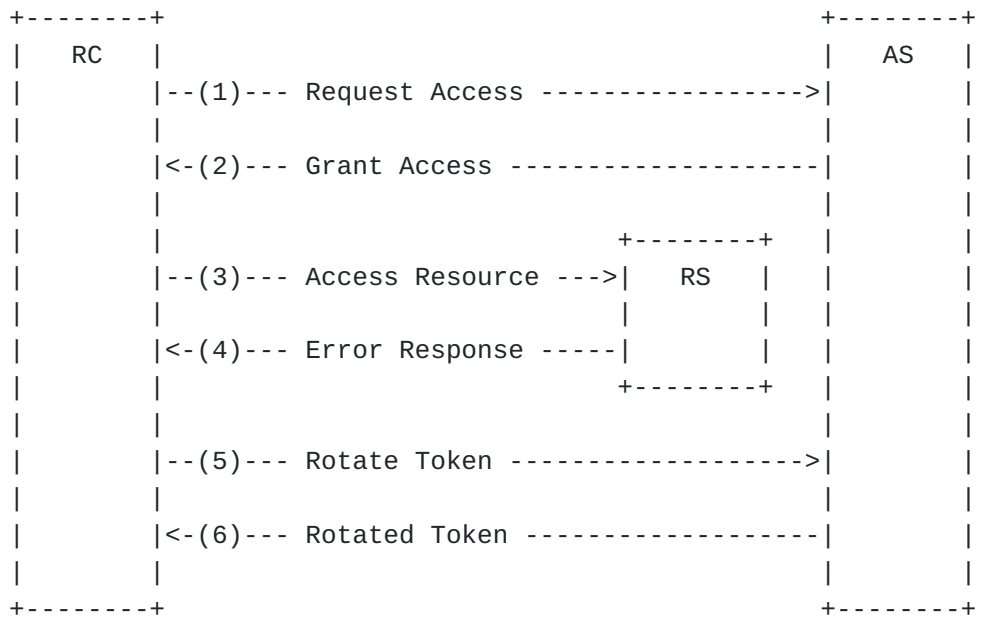
2. The AS determines that the request is been authorized, the AS grants access to the information in the form of access tokens (Section 3.2) and direct subject information (Section 3.4) to the RC.

An example set of protocol messages for this method can be found in Appendix D.

1.2.5. Refreshing an Expired Access Token

In this example flow, the RC receives an access token to access a resource server through some valid GNAP process. The RC uses that token at the RS for some time, but eventually the access token

expires. The RC then gets a new access token by rotating the expired access token at the AS using the token's management URL.



1. The RC requests access to the resource (Section 2).

2. The AS grants access to the resource (Section 3) with an access token (Section 3.2) usable at the RS. The access token response includes a token management URI.

3. The RC presents the token (Section 7) to the RS. The RS validates the token and returns an appropriate response for the API.

4. When the access token is expired, the RS responds to the RC with an error.

5. The RC calls the token management URI returned in (2) to rotate the access token (Section 6.1). The RC presents the access token as well as the appropriate key.

6. The AS validates the rotation request including the signature and keys presented in (5) and returns a new access token (Section 3.2.1). The response includes a new access token and can also include updated token management information, which the RC will store in place of the values returned in (2).

2. Requesting Access

To start a request, the client sends JSON [RFC8259] document with an object as its root. Each member of the request object represents a different aspect of the client's request.

A non-normative example of a grant request is below:

```
{
    "resources": [
        {
            "type": "photo-api",
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        },
        "dolphin-metadata"
    ],
    "key": {
        "proof": "jwsd",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
                    "n": "kOB5rR4Jv0GMeL...."
        }
    },
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.example.net/return/123455",
            "nonce": "LKLTI25DK82FX4T4QFZC"
        }
    },
    "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
    },
    "capabilities": ["ext1", "ext2"],
    "subject": {
        "sub_ids": ["iss-sub", "email"],
        "assertions": ["oidc_id_token"]
    }
}
```

The request MUST be sent as a JSON object in the body of the HTTP POST request with Content-Type application/json, unless otherwise specified by the signature mechanism.

2.1. Requesting Resources

If the client is requesting one or more access tokens for the purpose of accessing an API, the client MUST include a resources element. This element MUST be an array (for a single access token) or an object (for multiple access tokens), as described in the following sections.

2.1.1. Requesting a Single Access Token

When requesting a single access token, the client MUST send a resources element containing a JSON array. The elements of the JSON array represent rights of access that the client is requesting in the access token. The requested access is the sum of all elements within the array. These request elements MAY be sent by value as an object or by reference as a string. A single resources array MAY contain both object and string type resource requests.

The client declares what access it wants to associated with the resulting access token using objects that describe multiple dimensions of access. Each object contains a type property that determines the type of API that the client is calling. The value of this field is under the control of the AS and it MAY determine which other fields allowed in the object. While it is expected that many APIs will have its own properties, a set of common properties are defined here. Specific API implementations SHOULD NOT re-use these fields with different semantics or syntax.

[[Editor's note: this will align with OAuth 2 RAR, but the details of how it aligns are TBD]].

actions The types of actions the RC will take at the RS as an array of strings. The values of the strings are determined by the API being protected.

locations The location of the RS as an array of strings. These strings are typically URIs, and are determined by the API being protected.

datatypes Kinds of data available to the RC at the RS's API as an array of strings. The values of the strings are determined by the API being protected.

identifier A string identifier indicating a specific resource at the RS. The value of the string is determined by the API being protected.

The following non-normative example shows the use of both common and API-specific elements.

```
 "resources": [
     {
         "type": "photo-api",
         "actions": [
             "read",
             "write",
             "dolphin"
         ],
         "locations": [
             "https://server.example.net/",
             "https://resource.local/other"
         ],
         "datatypes": [
             "metadata",
             "images"
         ]
     },
     {
         "type": "financial-transaction",
         "actions": [
             "withdraw"
         ],
         "identifier": "account-14-32-32-3",
         "currency": "USD"
     }
 ]
```

2.1.2. Requesting Resources By Reference

Instead of sending an object describing the requested resource (Section 2.1.1), a client MAY send a string known to the AS or RS representing the access being requested. Each string SHOULD correspond to a specific expanded object representation at the AS.

[[Editor's note: we could describe more about how the expansion would work. For example, expand into an object where the value of the "type" field is the value of the string. Or we could leave it open and flexible, since it's really up to the AS/RS to interpret.]]

```
 "resources": [
     "read", "dolphin-metadata", "some other thing"
 ]
```

This value is opaque to the client and MAY be any valid JSON string, and therefore could include spaces, unicode characters, and properly escaped string sequences.

This functionality is similar in practice to OAuth 2's scope parameter [RFC6749], where a single string represents the set of access rights requested by the client. As such, the reference string could contain any valid OAuth 2 scope value as in Appendix D.2. Note that the reference string here is not bound to the same character restrictions as in OAuth 2's scope definition.

A single "resources" array MAY include both object-type and string-type resource items.

```
 "resources": [
     {
         "type": "photo-api",
         "actions": [
             "read",
             "write",
             "dolphin"
         ],
         "locations": [
             "https://server.example.net/",
             "https://resource.local/other"
         ],
         "datatypes": [
             "metadata",
             "images"
         ]
     },
     "read", "dolphin-metadata",
     {
         "type": "financial-transaction",
         "actions": [
             "withdraw"
         ],
         "identifier": "account-14-32-32-3",
         "currency": "USD"
     },
     "some other thing"
 ]
```

2.1.3. Requesting Multiple Access Tokens

When requesting multiple access tokens, the resources element is a JSON object. The names of the JSON object elements are token identifiers chosen by the client, and MAY be any valid string. The values of the JSON object are JSON arrays representing a single

access token request, as specified in requesting a single access token (Section 2.1.1).

The following non-normative example shows a request for two separate access tokens, token1 and token2.

```
 "resources": {
     "token1": [
       {
           "type": "photo-api",
           "actions": [
               "read",
               "write",
               "dolphin"
           ],
           "locations": [
               "https://server.example.net/",
               "https://resource.local/other"
           ],
           "datatypes": [
               "metadata",
               "images"
           ]
       },
       "dolphin-metadata"
   ],
   "token2": [
         {
             "type": "walrus-access",
             "actions": [
                 "foo",
                 "bar"
             ],
             "locations": [
                 "https://resource.other/"
             ],
             "datatypes": [
                 "data",
                 "pictures",
                 "walrus whiskers"
             ]
         }
     ]
 }
```

2.2. Requesting User Information

If the client is requesting information about the current user from the AS, it sends a subject element as a JSON object. This object MAY

contain the following fields (or additional fields defined in a registry TBD (Section 12)).

sub_ids An array of subject identifier subject types requested for the user, as defined by [I-D.ietf-secevent-subject-identifiers].

assertions An array of requested assertion formats defined by a registry TBD (Section 12).

```
"subject": {
   "sub_ids": [ "iss-sub", "email" ],
   "assertions": [ "oidc-id-token", "saml" ]
}
```

If the AS knows the identifier for the current user and has permission to do so [[editor's note: from the user's consent or data policy or ...]], the AS MAY return the user's information in its response (Section 3.4).

The "sub_ids" and "assertions" request fields are independent of each other, and a returned assertion MAY omit a requested subject identifier.

[[Editor's note: we're potentially conflating these two fields in the same structure, so perhaps these should be split. There's also a difference between user information and authentication event information.]]

2.3. Identifying the Client Key

When sending an initial request to the AS, the client MUST identify itself by including the key field in the request and by signing the request as described in Section 8. This key MAY be sent by value or by reference.

When sent by value, the key MUST be a public key in at least one supported format and MUST contain a proof property that matches the proofing mechanism used in the request. If the key is sent in multiple formats, all the keys MUST be the same. The key presented in this field MUST be the key used to sign the request.

proof The form of proof that the RC will use when presenting the key to the AS. The valid values of this field and the processing requirements for each are detailed in Section 8. This field is REQUIRED.

jwk Value of the public key as a JSON Web Key. MUST contain an "alg" field which is used to validate the signature. MUST contain the "kid" field to identify the key in the signed object.

cert
: PEM serialized value of the certificate used to sign the request, with optional internal whitespace.

cert#256 The certificate thumbprint calculated as per OAuth-MTLS [RFC8705] in base64 URL encoding.

Additional key types are defined in a registry TBD (Section 12).

[[Editor's note: we will eventually want to have fetchable keys, I would guess. Things like DID for key identification are going to be important.]]

This non-normative example shows a single key presented in multiple formats using a single proofing mechanism.

```
 "key": {
     "proof": "httpsig",
     "jwk": {
                 "kty": "RSA",
                 "e": "AQAB",
                 "kid": "xyz-1",
                 "alg": "RS256",
                 "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_JtffXyaSx8xY..
     },
     "cert": "MIIEHDCCAwSgAwIBAgIBATANBgkqhkiG9w0BAQsFA..."
 }
```

The RC MUST prove possession of any presented key by the proof mechanism associated with the key in the request. Proof types are defined in a registry TBD (Section 12) and an initial set are of methods are described in Section 8. Continuation requests (Section 5) MUST use the same key and proof method as the initial request.

[[Editor's note: additional client attestation frameworks will eventually need to be addressed here beyond the presentation of the key. For example, the organization the client represents, or a family of client software deployed in a cluster, or the posture of the device the client is installed on. These all need to be separable from the client's key and the key identifier.]]

2.3.1. Authenticating the Client

If the presented key is known to the AS and is associated with a single instance of a client, the process of presenting a key and proving possession of that key is usually sufficient to authenticate the client to the AS. The AS MAY associate policies with the client software identified by this key, such as limiting which resources can be requested and which interaction methods can be used. For example, only specific clients with certain known keys might be

trusted with access tokens without the AS interacting directly with the user as in Appendix D.

The presentation of a key is of vital importance to the protocol as it allows the AS to strongly associate multiple requests from the same RC with each other. This value exists whether the AS knows the key ahead of time or not, and as such the AS MAY allow for clients to make requests with unknown keys. This pattern allows for ephemeral clients, such as single-page applications, and many-instance clients, such as mobile applications, to generate their own key pairs and use them within the protocol without having to go through a separate registration step. The AS MAY limit which capabilities are made available to clients with unknown keys. For example, the AS could have a policy saying that only previously-registered clients can request particular resources.

2.3.2. Identifying the Client Key By Reference

If the client has a reference for its key, the client MAY send that reference handle as a string. The format of this string is opaque to the client.

```
{
  "key": "7C7C4AZ9KHRS6X63AJAO"
}
```

If the key is passed by reference, the proofing mechanism associated with that key reference MUST also be used by the client, as described in Section 8.

If the AS does not recognize the key reference handle, the request MUST be rejected with an error.

If the client identifies its key by reference, the referenced key MAY be a symmetric key known to the AS. The client MUST NOT send a symmetric key by value, as doing so would be a security violation.

[[Editor's note: In many ways, passing a key identifier by reference is analogous to OAuth 2's "client_id" parameter [RFC6749], especially when coupled with a confidential client's authentication process. See Appendix D.2 for an example.]]

2.4. Identifying the User

If the client knows the identity of the current user or one or more identifiers for the user, the client MAY send that information to the AS in the "user" field. The client MAY pass this information by value or by reference.

sub_ids

An array of subject identifiers for the user, as defined by [I-D.ietf-secevent-subject-identifiers].

assertions An object containing assertions as values keyed on the assertion type defined by a registry TBD (Section 12). [[Editor's note: should this be an array of objects with internal typing like the sub_ids? Do we expect more than one assertion per user anyway?]]

```
"user": {
   "sub_ids": [ {
     "subject_type": "email",
     "email": "user@example.com"
   } ],
   "assertions": {
     "oidc_id_token": "eyj..."
   }
}
```

Subject identifiers are hints to the AS in determining the current user and MUST NOT be taken as declarative statements that a particular user is present at the client. Assertions SHOULD be validated by the AS. [[editor's note: assertion validation is extremely specific to the kind of assertion in place]]

If the identified user does not match the user present at the AS during an interaction step, the AS SHOULD reject the request.

[[Editor's note: we're potentially conflating identification (sub_ids) and provable presence (assertions and a trusted reference handle) in the same structure, so perhaps these should be split.]]

Additional user assertion formats are defined in a registry TBD (Section 12). [[Editor's note: probably the same registry as requesting formats to keep them aligned.]]

If the AS trusts the client to present user information, it MAY decide, based on its policy, to skip interaction with the user, even if the client provides one or more interaction capabilities.

2.4.1. Identifying the User by Reference

If the client has a reference for the current user at this AS, the client MAY pass that reference as a string. The format of this string is opaque to the client.

```
"user": "XUT2MFM1XBIKJKSDU8QM"
```

User reference identifiers are not intended to be human-readable user identifiers or machine-readable verifiable assertions. For either of these, use the regular user request instead.

If the AS does not recognize the user reference, it MUST return an error.

2.5. Interacting with the User

If the client is capable of driving interaction with the user, the client SHOULD declare the means that it can interact using the "interact" field. This field is a JSON object with keys that declare different interaction capabilities. A client MUST NOT declare an interaction capability it does not support.

The client MAY send multiple capabilities in the same request. There is no preference order specified in this request. An AS MAY respond to any, all, or none of the presented interaction capabilities (Section 3.3) in a request, depending on its capabilities and what is allowed to fulfill the request.

The following sections detail requests for interaction capabilities. Additional interaction capabilities are defined in a registry TBD (Section 12).

[[Editor's note: there need to be more examples (Appendix C) that knit together the interaction capabilities into common flows, like an authz-code equivalent. But it's important for the protocol design that these are separate pieces to allow such knitting to take place.]]

```
 "interact": {
     "redirect": true,
     "user_code": true,
     "callback": {
         "method": "redirect",
         "uri": "https://client.example.net/return/123455",
         "nonce": "LKLTI25DK82FX4T4QFZC"
     }
 }
```

If the RC does not provide a suitable interaction mechanism, the AS cannot contact the RO asynchronously, and the AS determines that interaction is required, then the AS SHOULD return an error since the RC will be unable to complete the request without authorization.

2.5.1. Redirect to an Arbitrary URL

If the client is capable of directing the user to a URL defined by the AS at runtime, the client indicates this by sending the

"redirect" field with the boolean value "true". The means by which the client will activate this URL is out of scope of this specification, but common methods include an HTTP redirect, launching a browser on the user's device, providing a scannable image encoding, and printing out a URL to an interactive console.

```
"interact": {
   "redirect": true
}
```

If this interaction capability is supported for this client and request, the AS returns a redirect interaction response Section 3.3.1.

2.5.1.1. Redirect to an Arbitrary Shortened URL

If the client would prefer to redirect to a shortened URL defined by the AS at runtime, the client indicates this by sending the "redirect" field with an integer indicating the maximum character length of the returned URL. The AS MAY use this value to decide whether to return a shortened form of the response URL. If the AS cannot shorten its response URL enough to fit in the requested size, the AS SHOULD return an error. [[Editor's note: Or maybe just ignore this part of the interaction request?]]

The means by which the client will activate this URL is out of scope of this specification, but common methods include an HTTP redirect, launching a browser on the user's device, providing a scannable image encoding, and printing out a URL to an interactive console for the user to copy and paste into a browser.

```
"interact": {
   "redirect": 255
}
```

If this interaction capability is supported for this client and request, the AS returns a redirect interaction response with short URL Section 3.3.1.

2.5.2. Open an Application-specific URL

If the client can open a URL associated with an application on the user's device, the client indicates this by sending the "app" field with boolean value "true". The means by which the client determines the application to open with this URL are out of scope of this specification.

```
"interact": {
   "app": true
}
```

If this interaction capability is supported for this client and request, the AS returns an app interaction response with an app URL payload Section 3.3.2.

[[Editor's note: this is similar to the "redirect" above today as most apps use captured URLs, but there seems to be a desire for splitting the web-based interaction and app-based interaction into different URIs. There's also the possibility of wanting more in the payload than can be reasonably put into the URL, or at least having separate payloads.]]

2.5.3. Receive a Callback After Interaction

If the client is capable of receiving a message from the AS indicating that the user has completed their interaction, the client indicates this by sending the "callback" field. The value of this field is an object containing the following members.

uri REQUIRED. Indicates the URI to send the RO to after interaction. This URI MAY be unique per request and MUST be hosted by or accessible by the RC. This URI MUST NOT contain any fragment component. This URI MUST be protected by HTTPS, be hosted on a server local to the user's browser ("localhost"), or use an application-specific URI scheme. If the RC needs any state information to tie to the front channel interaction response, it MUST encode that into the callback URI. The allowable URIs and URI patterns MAY be restricted by the AS based on the RC's presented key information. The callback URI SHOULD be presented to the RO during the interaction phase before redirect.

nonce REQUIRED. Unique value to be used in the calculation of the "hash" query parameter sent to the callback URL, must be sufficiently random to be unguessable by an attacker. MUST be generated by the RC as a unique value for this request.

method REQUIRED. The callback method that the AS will use to contact the client. Valid values include redirect Section 2.5.3.1 and push Section 2.5.3.2, with other values defined by a registry TBD (Section 12).

hash_method OPTIONAL. The hash calculation mechanism to be used for the callback hash in Section 4.4.3. Can be one of sha3 or sha2. If absent, the default value is sha3. [[Editor's note: This should be expandable via a registry of cryptographic options, and it would be good if we didn't define our own identifiers here. See also note about cryptographic functions in Section 4.4.3.]]

```
"interact": {
    "callback": {
       "method": "redirect",
       "uri": "https://client.example.net/return/123455",
       "nonce": "LKLTI25DK82FX4T4QFZC"
    }
}
```

If this interaction capability is supported for this client and request, the AS returns a nonce for use in validating the callback response (Section 3.3.3). Requests to the callback URI MUST be processed as described in Section 4.4, and the AS MUST require presentation of an interaction callback reference as described in Section 5.1.

Note that the means by which the user arrives at the AS is declared separately from the user's return using this callback mechanism.

2.5.3.1. Receive an HTTP Callback Through the Browser

A callback method value of redirect indicates that the client will expect a call from the user's browser using the HTTP method GET as described in Section 4.4.1.

```
"interact": {
    "callback": {
       "method": "redirect",
       "uri": "https://client.example.net/return/123455",
       "nonce": "LKLTI25DK82FX4T4QFZC"
    }
}
```

Requests to the callback URI MUST be processed as described in Section 4.4.1.

Since the incoming request to the callback URL is from the user's browser, the client MUST require the user to be present on the connection.

2.5.3.2. Receive an HTTP Direct Callback

A callback method value of push indicates that the client will expect a call from the AS directly using the HTTP method POST as described in Section 4.4.2.

```
"interact": {
    "callback": {
       "method": "redirect",
       "uri": "https://client.example.net/return/123455",
       "nonce": "LKLTI25DK82FX4T4QFZC"
    }
}
```

Requests to the pushback URI MUST be processed as described in Section 4.4.2.

Since the incoming request to the pushback URL is from the AS and not from the user's browser, the client MUST NOT require the user to be present.

2.5.4. Display a Short User Code

If the client is capable of displaying or otherwise communicating a short, human-entered code to the user, the client indicates this by sending the "user_code" field with the boolean value "true". This code is to be entered at a static URL that does not change at runtime.

```
"interact": {
    "user_code": true
}
```

If this interaction capability is supported for this client and request, the AS returns a user code and interaction URL as specified in Section 4.2.

2.5.5. Extending Interaction Capabilities

Additional interaction capabilities are defined in a registry TBD (Section 12).

[[Editor's note: we should have guidance in here about how to define other interaction capabilities. There's already interest in defining message-based protocols and challenge-response protocols, for example.]]

2.6. Providing Displayable Client Information

If the client has additional information to display to the user during any interactions at the AS, it MAY send that information in the "display" field. This field is a JSON object that declares information to present to the user during any interactive sequences.

name Display name of the RC software

uri
: User-facing web page of the RC software

logo_uri Display image to represent the RC software

```
 "display": {
     "name": "My Client Display Name",
     "uri": "https://example.net/client"
 }
```

Additional display fields are defined by a registry TBD (Section 12).

The AS SHOULD use these values during interaction with the user. The AS MAY restrict display values to specific clients, as identified by their keys in Section 2.3.

[[Editor's note: this might make sense to combine with the "key" field, but some classes of more dynamic client vary those fields separately from the key material. We should also consider things like signed statements for client attestation, but that might fit better into a different top-level field instead of this "display" field.]]

2.7. Declaring Client Capabilities

If the client supports extension capabilities, it MAY present them to the AS in the "capabilities" field. This field is an array of strings representing specific extensions and capabilities, as defined by a registry TBD (Section 12).

```
"capabilities": ["ext1", "ext2"]
```

2.8. Referencing an Existing Grant Request

If the client has a reference handle from a previously granted request, it MAY send that reference in the "reference" field. This field is a single string.

```
"existing_grant": "80UPRY5NM33OMUKMKSKU"
```

The AS MUST dereference the grant associated with the reference and process this request in the context of the referenced one.

[[Editor's note: this basic capability is to allow for both step-up authorization and downscoped authorization, but by explicitly creating a new request and not modifying an existing one. What's the best guidance for how an AS should process this?]]

2.9. Requesting OpenID Connect Claims

If the client and AS both support OpenID Connect's claims query language as defined in [OIDC] Section 5.5, the client sends the value of the OpenID Connect claims authorization request parameter as a JSON object under the name oidc_claims.

```
"oidc_claims": {
        "id_token" : {
            "email"          : { "essential" : true },
            "email_verified" : { "essential" : true }
        },
        "userinfo" : {
            "name"           : { "essential" : true },
            "picture"        : null
        }
}
```

The contents of the oidc_claims parameter have the same semantics as they do in OpenID Connect, including all extensions such as [OIDC4IA]. The AS MUST process the claims object in the same way that it would with an OAuth 2 based authorization request.

Note that because this is an independent query object, the oidc_claims value can augment or alter other portions of the request, namely the resources and subject fields. This query language uses the fields in the top level of the object to indicate the target for any requested claims. For instance, the userinfo target indicates that an access token would grant access to the given claims at the UserInfo Endpoint, while the id_token target indicates that the claims would be returned in an ID Token as described in Section 3.4.

[[Editor's note: I'm not a fan of GNAP defining how OIDC would work and would rather that work be done by the OIDF. However, I think it is important for discussion to see this kind of thing in context with the rest of the protocol, for now.]]

2.10. Extending The Grant Request

The request object MAY be extended by registering new items in a registry TBD (Section 12). Extensions SHOULD be orthogonal to other parameters. Extensions MUST document any aspects where the

[[Editor's note: we should have more guidance and examples on what possible top-level extensions would look like. Things like an OIDC "claims" request or a VC query, for example.]]

3. Grant Response

In response to a client's request, the AS responds with a JSON object as the HTTP entity body.

In this example, the AS is returning an interaction URL (Section 3.3.1), a callback nonce (Section 3.3.3), and a continuation handle (Section 3.1).

```
{
    "interact": {
        "redirect": "https://server.example.com/interact/4CF492MLVMSW9MK
         "callback": "MBDOFXG4Y5CVJCX821LH"
    },
    "continue": {
        "handle": "80UPRY5NM33OMUKMKSKU",
        "uri": "https://server.example.com/tx"
    }
}
```

In this example, the AS is returning an access token (Section 3.2.1), a continuation handle (Section 3.1), and a subject identifier (Section 3.4).

```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "proof": "bearer",
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7O
    },
    "continue": {
        "handle": "80UPRY5NM33OMUKMKSKU",
        "uri": "https://server.example.com/continue"
    },
    "subject": {
        "sub_ids": [ {
           "subject_type": "email",
           "email": "user@example.com",
        } ]
    }
}
```

3.1. Request Continuation Handle

If the AS determines that the request can be continued with additional requests, it responds with the "continue" field. This field contains a JSON object with the following properties.

handle REQUIRED. A unique reference for the grant request.

uri
: REQUIRED. The URI at which the client can make continuation requests. This URI MAY vary per client or ongoing request, or MAY be stable at the AS.

wait : RECOMMENDED. The amount of time in integer seconds the client SHOULD wait after receiving this continuation handle and calling the URI.

expires_in : OPTIONAL. The number of seconds in which the handle will expire. The client MUST NOT use the handle past this time. The handle MAY be revoked at any point prior to its expiration.

```
{
 "continue": {
     "handle": "80UPRY5NM33OMUKMKSKU",
     "uri": "https://server.example.com/continue",
     "wait": 60
 }
}
```

The client can use the values of this field as described in Section 5.

This field SHOULD be returned when interaction is expected, to allow the client to follow up after interaction has been concluded.

3.2. Access Tokens

If the AS has successfully granted one or more access tokens, it responds with one of these fields. The AS MUST NOT respond with both fields.

[[Editor's note: I really don't like the dichotomy between "access_token" and "multiple_access_tokens" and their being mutually exclusive, and I think we should design away from this pattern toward something less error-prone.]]

3.2.1. Single Access Token

If the client has requested a single access token and the AS has granted that access token, the AS responds with the "access_token" field. The value of this field is an object with the following properties.

value : REQUIRED. The value of the access token as a string. The value is opaque to the client. The value SHOULD be limited to ASCII characters to facilitate transmission over HTTP headers and elements without additional encoding.

proof

REQUIRED. The proofing presentation mechanism used for presenting this access token to an RS. See the section on using access tokens (Section 7) for details on possible values to this field and their requirements.

manage OPTIONAL. The management URI for this access token. If provided, the client MAY manage its access token as described in managing an access token lifecycle (Section 6). This URI MUST NOT include the access token value and MAY be different for each access token.

resources OPTIONAL. A description of the rights associated with this access token, as defined in requesting resource access (Section 3.2.1). If included, this MUST reflect the rights associated with the issued access token. These rights MAY vary from what was requested by the client.

expires_in OPTIONAL. The number of seconds in which the access will expire. The client MUST NOT use the access token past this time. The access token MAY be revoked at any point prior to its expiration.

key The key that the token is bound to, REQUIRED if the token is sender-constrained. The key MUST be in a format described in Section 2.3. [[Editor's note: this isn't quite right, since the request section includes a "proof" field that we already have here. A possible solution would be to only have a "key" field as defined above and its absence indicates a bearer token?]]

```
"access_token": {
    "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
    "proof": "bearer",
    "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7O
    "resources": [
        {
            "type": "photo-api",
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        },
        "read", "dolphin-metadata"
    ]
}
```

3.2.2. Multiple Access Tokens

If the client has requested multiple access tokens and the AS has granted at least one of them, the AS responds with the "multiple_access_tokens" field. The value of this field is a JSON object, and the property names correspond to the token identifiers chosen by the client in the multiple access token request (Section 2.1.3). The values of the properties of this object are access tokens as described in Section 3.2.1.

```
"multiple_access_tokens": {
    "token1": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "proof": "bearer",
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6
    },
    "token2": {
        "value": "UFGLO2FDAFG7VGZZPJ3IZEMN21EVU71FHCARP4J1",
        "proof": "bearer"
    }
}
```

Each access token corresponds to the named resources arrays in the client's request. The AS MAY not issue one or more of the requested

access tokens. In such cases all of the issued access tokens are included without the omitted token. The multiple access token response MUST be used when multiple access tokens are requested, even if only one access token is issued.

If the client requested a single access token (Section 2.1.1), the AS MUST NOT respond with multiple access tokens.

Each access token MAY have different proofing mechanisms. If used, each access token MUST have different management URIs.

3.3. Interaction Capabilities

If the client has indicated a capability to interact with the user in its request (Section 2.5), and the AS has determined that interaction is both supported and necessary, the AS responds to the client with any of the following values in the interact field of the response. There is no preference order for interaction capabilities in the response, and it is up to the client to determine which ones to use.

The AS MUST NOT respond with any interaction capability that the client did not indicate in its request.

3.3.1. Redirection to an arbitrary URL

If the client indicates that it can redirect to an arbitrary URL (Section 2.5.1) and the AS supports this capability for the client's request, the AS responds with the "redirect" field, which is a string containing the URL to direct the user to. This URL MUST be unique for the request and MUST NOT contain any security-sensitive information.

```
"interact": {
    "redirect": "https://server.example.com/interact/4CF492MLVMSW9MK
}
```

The client sends the user to the URL to interact with the AS. The client MUST NOT alter the URL in any way. The means for the client to send the user to this URL is out of scope of this specification, but common methods include an HTTP redirect, launching the system browser, displaying a scannable code, or printing out the URL in an interactive console.

3.3.2. Launch of an application URL

If the client indicates that it can launch an application URL (Section 2.5.2) and the AS supports this capability for the client's request, the AS responds with the "app" field, which is a string containing the URL to direct the user to. This URL MUST be unique

for the request and MUST NOT contain any security-sensitive information.

```
 "interact": {
     "app": "https://app.example.com/launch?tx=4CF492MLV"
 }
```

The client launches the URL as appropriate on its platform, and the means for the client to launch this URL is out of scope of this specification. The client MUST NOT alter the URL in any way. The client MAY attempt to detect if an installed application will service the URL being sent.

[[Editor's note: This will probably need to be expanded to an object to account for other parameters needed in app2app use cases, like addresses for distributed storage systems, server keys, and the like. Details TBD as people build this out.]]

3.3.3. Callback to a Client URL

If the client indicates that it can receive a post-interaction callback on a URL (Section 2.5.3) and the AS supports this capability for the client's request, the AS responds with a "callback" field containing a nonce that the client will use in validating the callback as defined in Section 4.4.1.

```
 "interact": {
     "callback": "MBDOFXG4Y5CVJCX821LH"
 }
```

When the user completes interaction at the AS, the AS MUST call the client's callback URL using the method indicated in the callback request (Section 2.5.3) as described in Section 4.4.1.

If the AS returns a "callback" nonce, the client MUST NOT continue a grant request before it receives the associated interaction reference on the callback URI.

3.3.4. Display of a Short User Code

If the client indicates that it can display a short user-typeable code (Section 2.5.4) and the AS supports this capability for the client's request, the AS responds with a "user_code" field. This field is an object that contains the following members.

code REQUIRED. A unique short code that the user can type into an authorization server. This string MUST be case-insensitive, MUST consist of only easily typeable characters (such as letters or numbers). The time in which this code will be accepted SHOULD be

short lived, such as several minutes. It is RECOMMENDED that this code be no more than eight characters in length.

url RECOMMENDED. The interaction URL that the RC will direct the RO to. This URL MUST be stable at the AS such that clients can be statically configured with it.

```
 "interact": {
     "user_code": {
         "code": "A1BC-3DFF",
         "url": "https://srv.ex/device"
     }
 }
```

The client MUST communicate the "code" to the user in some fashion, such as displaying it on a screen or reading it out audibly. The client SHOULD also communicate the URL if possible.

The code is a one-time-use credential that the AS uses to identify the pending request from the RC. When the user enters this code into the AS, the AS MUST determine the pending request that it was associated with. If the AS does not recognize the entered code, the AS MUST display an error to the user.

As this interaction capability is designed to facilitate interaction via a secondary device, it is not expected that the client redirect the user to the URL given here at runtime. Consequently, the URL needs to be stable enough that a client could be statically configured with it, perhaps referring the user to the URL via documentation instead of through an interactive means. If the client is capable of communicating an arbitrary URL to the user, such as through a scannable code, the client can use the "redirect" (Section 2.5.1) capability for this purpose.

3.3.5. Extending Interaction Capability Responses

Extensions to this specification can define new interaction capability responses in a registry TBD (Section 12).

3.4. Returning User Information

If information about the current user is requested and the AS grants the client access to that data, the AS returns the approved information in the "subject" response field. This field is an object with the following OPTIONAL properties.

sub_ids An array of subject identifiers for the user, as defined by [I-D.ietf-secevent-subject-identifiers]. [[Editor's note: privacy considerations are needed around returning identifiers.]]

assertions
: An object containing assertions as values keyed on the assertion type defined by a registry TBD (Section 12). [[Editor's note: should this be an array of objects with internal typing like the sub_ids? Do we expect more than one assertion per user anyway?]]

updated_at
: Timestamp in integer seconds indicating when the identified account was last updated. The client MAY use this value to determine if it needs to request updated profile information through an identity API.

```
"subject": {
   "sub_ids": [ {
     "subject_type": "email",
     "email": "user@example.com",
   } ],
   "assertions": {
     "oidc_id_token": "eyj..."
   }
}
```

Extensions to this specification MAY define additional response properties in a registry TBD (Section 12).

3.5. Returning Dynamically-bound Reference Handles

Many parts of the client's request can be passed as either a value or a reference. Some of these references, such as for the client's keys or the resources, can sometimes be managed statically through an admin console or developer portal provided by the AS or RS. If desired, the AS MAY also generate and return some of these references dynamically to the client in its response to facilitate multiple interactions with the same software. The client SHOULD use these references in future requests in lieu of sending the associated data value. These handles are intended to be used on future requests.

Dynamically generated handles are string values that MUST be protected by the client as secrets. Handle values MUST be unguessable and MUST NOT contain any sensitive information. Handle values are opaque to the client. [[Editor's note: these used to be objects to allow for expansion to future elements, like a management URI or different presentation types or expiration, but those weren't used in practice. Is that desirable anymore or is collapsing them like this the right direction?]]

All dynamically generated handles are returned as fields in the root JSON object of the response. This specification defines the

following dynamic handle returns, additional handles can be defined in a registry TBD (Section 12).

key_handle A value used to represent the information in the key object that the client can use in a future request, as described in Section 2.3.2.

user_handle A value used to represent the current user. The client can use in a future request, as described in Section 2.4.1.

This non-normative example shows two handles along side an issued access token.

```
{
    "user_handle": "XUT2MFM1XBIKJKSDU8QM",
    "key_handle": "7C7C4AZ9KHRS6X63AJAO",
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "proof": "bearer"
    }
}
```

3.6. Error response

If the AS determines that the request cannot be issued for any reason, it responds to the RC with an error message.

error The error code.

```
{

  "error": "user_denied"

}
```

The error code is one of the following, with additional values available in a registry TBD (Section 12):

user_denied The RO denied the request.

too_fast The RC did not respect the timeout in the wait response.

unknown_handle The request referenced an unknown handle.

[[Editor's note: I think we will need a more robust error mechanism, and we need to be more clear about what error states are allowed in what circumstances. Additionally, is the "error" parameter exclusive with others in the return?]]

3.7. Extending the Response

Extensions to this specification MAY define additional fields for the grant response in a registry TBD (Section 12).

[[Editor's note: what guidance should we give to designers on this?]]

4. Interaction at the AS

If the client indicates that it is capable of driving interaction with the user in its request (Section 2.5), and the AS determines that interaction is required and responds to one or more of the client's interaction capabilities, the client SHOULD initiate one of the returned interaction capabilities in the response (Section 3.3).

When the RO is interacting with the AS, the AS MAY perform whatever actions it sees fit, including but not limited to:

- authenticate the user as RO
- gather consent and authorization from the RO for access to requested resources or the
- allow the RO to modify the parameters of the request (such as disallowing some requested resources or specifying an account or record)

[[Editor's note: there are some privacy and security considerations here but for the most part we don't want to be overly prescriptive about the UX, I think.]]

4.1. Interaction at a Redirected URI

When the user is directed to the AS through the "redirect" (Section 3.3.1) capability, the AS can interact with the user through their web browser to authenticate the user as an RO and gather their consent. Note that since the client does not add any parameters to the URL, the AS MUST determine the grant request being referenced from the URL value itself. If the URL cannot be associated with a currently active request, the AS MUST display an error to the user and MUST NOT attempt to redirect the user back to any client.

The interaction URL MUST be reachable from the RO's browser, though note that the RO MAY open the URL on a separate device from the RC itself. The interaction URL MUST be accessible from an HTTP GET request, and MUST be protected by HTTPS or equivalent means.

4.2. Interaction at the User Code URI

When the user is directed to the AS through the "user_code" (Section 3.3.4) capability, the AS can interact with the user through their web browser to collect the user code, authenticate the user as an RO, and gather their consent. Note that since the URL itself is static, the AS MUST determine the grant request being referenced from the user code value itself. If the user code cannot be associated with a currently active request, the AS MUST display an error to the user and MUST NOT attempt to redirect the user back to any client.

The user code URL MUST be reachable from the RO's browser, though note that the RO MAY open the URL on a separate device from the RC itself. The user code URL MUST be accessible from an HTTP GET request, and MUST be protected by HTTPS or equivalent means.

4.3. Interaction through an Application URI

When the user successfully launches an application through the "app" capability (Section 3.3.2), the AS interacts with the user through that application to authenticate the user as the RO and gather their consent. The details of this interaction are out of scope for this specification.

[[Editor's note: Should we have anything to say about an app sending information to a back-end to get details on the pending request?]]

4.4. Post-Interaction Completion

Upon completing an interaction with the user, if a "callback" (Section 3.3.3) capability is available with the current request, the AS MUST follow the appropriate method at the end of interaction to allow the client to continue. If neither capability is available, the AS SHOULD instruct the user to return to their client software upon completion. Note that these steps still take place in most error cases, such as when the user has denied access. This allows the client to potentially recover from the error state without restarting.

[[Editor's note: there might be some other kind of push-based notification or callback that the client can use, or an out-of-band non-HTTP protocol. The AS would know about this if supported and used, but the guidance here should be written in such a way as to not be too restrictive in the next steps that it can take. Still, it's important that the AS not expect or even allow clients to poll if the client has stated it can take a callback of some form, otherwise that sets up a potential session fixation attack vector that the client is trying to and able to avoid.]]

The AS MUST calculate a hash value as described in Section 4.4.3. The client will use this value to validate the return call from the AS.

The AS MUST create an interaction reference and associate that reference with the current interaction and the underlying pending request. This value MUST be sufficiently random so as not to be guessable by an attacker.

The AS then MUST send the hash and interaction reference based on the interaction finalization capability as described in the following sections.

4.4.1. Completing Interaction with a Callback URI

When using the "callback" interaction capability (Section 3.3.3) with the redirect method, the AS signals to the client that interaction is complete and the request can be continued by directing the user (in their browser) back to the client's callback URL sent in the callback request (Section 2.5.3.1).

The AS secures this callback by adding the hash and interaction reference as query parameters to the client's callback URL.

hash REQUIRED. The interaction hash value as described in Section 4.4.3.

interact_ref REQUIRED. The interaction reference generated for this interaction.

The means of directing the user to this URL are outside the scope of this specification, but common options include redirecting the user from a web page and launching the system browser with the target URL.

```
https://client.example.net/return/123455
  ?hash=p28jsq0Y2KK3WS__a42tavNC64ldGTBroywsWxT4md_jZQ1R2HZT8BOWYHcLmObM
  &interact_ref=4IFWWIKYBC2PQ6U56NL1
```

When receiving the request, the client MUST parse the query parameters to calculate and validate the hash value as described in Section 4.4.3. If the hash validates, the client sends a continuation request to the AS as described in Section 5.1 using the interaction reference value received here.

4.4.2. Completing Interaction with a Pushback URI

When using the "callback" interaction capability (Section 3.3.3) with the push method, the AS signals to the client that interaction is complete and the request can be continued by sending an HTTP POST

request to the client's callback URL sent in the callback request (Section 2.5.3.2).

The entity message body is a JSON object consisting of the following two elements:

hash REQUIRED. The interaction hash value as described in Section 4.4.3.

interact_ref REQUIRED. The interaction reference generated for this interaction.

```
POST /push/554321 HTTP/1.1
Host: client.example.net
Content-Type: application/json

{
  "hash": "p28jsq0Y2KK3WS__a42tavNC64ldGTBroywsWxT4md_jZQ1R2HZT8BOWYHcLm
  "interact_ref": "4IFWWIKYBC2PQ6U56NL1"
}
```

When receiving the request, the client MUST parse the JSON object and validate the hash value as described in Section 4.4.3. If the hash validates, the client sends a continuation request to the AS as described in Section 5.1 using the interaction reference value received here.

4.4.3. Calculating the interaction hash

The "hash" parameter in the request to the client's callback URL ties the front channel response to an ongoing request by using values known only to the parties involved. This prevents several kinds of session fixation attacks against the client.

To calculate the "hash" value, the party doing the calculation first takes the "nonce" value sent by the RC in the interaction section of the initial request (Section 2.5.3), the AS's nonce value from the callback response (Section 3.3.3), and the "interact_ref" sent to the client's callback URL. These three values are concatenated to each other in this order using a single newline character as a separator between the fields. There is no padding or whitespace before or after any of the lines, and no trailing newline character.

```
VJLO6A4CAYLBXHTR0KRO
MBDOFXG4Y5CVJCX821LH
4IFWWIKYBC2PQ6U56NL1
```

The party then hashes this string with the appropriate algorithm based on the "hash_method" parameter of the "callback". If the

"hash_method" value is not present in the RC's request, the algorithm defaults to "sha3".

[[Editor's note: these hash algorithms should be pluggable, and ideally we shouldn't redefine yet another crypto registry for this purpose, but I'm not convinced an appropriate one already exists. Furthermore, we should be following best practices here whether it's a plain hash, a keyed MAC, an HMAC, or some other form of cryptographic function. I'm not sure what the defaults and options ought to be, but SHA512 and SHA3 were picked based on what was available to early developers.]]

4.4.3.1. SHA3

The "sha3" hash method consists of hashing the input string with the 512-bit SHA3 algorithm. The byte array is then encoded using URL Safe Base64 with no padding. The resulting string is the hash value.

```
p28jsq0Y2KK3WS__a42tavNC64ldGTBroywsWxT4md_jZQ1R2HZT8BOWYHcLmObM7XHPAdJz
```

4.4.3.2. SHA2

The "sha2" hash method consists of hashing the input string with the 512-bit SHA2 algorithm. The byte array is then encoded using URL Safe Base64 with no padding. The resulting string is the hash value.

```
62SbcD3Xs7L40rjgALA-ymQujoh2LB2hPJyX9vlcr1H6ecChZ8BNKkG_HrOKP_Bpj84rh4mC
```

5. Continuing a Grant Request

If the client receives a continuation element in its response Section 3.1, the client can make an HTTP POST call to the continuation URI with a JSON object. The client MUST send the handle reference from the continuation element in its request as a top-level JSON parameter.

```
{
  "handle": "tghji76ytghj9876tghjko987yh"
}
```

The client MAY include other parameters as described here or as defined a registry TBD (Section 12).

[[Editor's note: We probably want to allow other parameters, like modifying the resources requested or providing more user information. We'll certainly have some kinds of specific challenge-response protocols as there's already been interest in that kind of thing, and the continuation request is the place where that would fit.]]

If a "wait" parameter was included in the continuation response, the client MUST NOT call the continuation URI prior to waiting the number of seconds indicated. If no "wait" period is indicated, the client SHOULD wait at least 5 seconds [[Editor's note: what's a reasonable amount of time so as not to DOS the server??]].

The response from the AS is a JSON object and MAY contain any of the elements described in Section 3, with some variations:

If the AS determines that the client can make a further continuation request, the AS MUST include a new "continue" response element (Section 3.1). The returned handle value MUST NOT be the same as that used to make the continuation request, and the continuation URI MAY remain the same. If the AS does not return a new "continue" response element, the client MUST NOT make an additional continuation request. If a client does so, the AS MUST return an error.

If the AS determines that the client still needs to drive interaction with the user, the AS MAY return appropriate responses for any of the interaction mechanisms (Section 3.3) the client indicated in its initial request (Section 2.5). Unique values such as interaction URIs and nonces SHOULD be re-generated and not re-used.

The client MUST present proof of the same key identified in the initial request (Section 2.3) by signing the request as described in Section 8. This requirement is in place whether or not the AS had previously registered the client's key as described in Section 2.3.1.

5.1. Continuing after a Finalized Interaction

If the client has received an interaction reference from a "callback" (Section 4.4.1) message, the client MUST include the "interaction_ref" in its continuation request. The client MUST validate the hash before making the continuation request, but note that the client does not send the hash back to the AS in the request.

```
{
  "handle": "tghji76ytghj9876tghjko987yh",
  "interact_ref": "4IFWWIKYBC2PQ6U56NL1"
}
```

5.2. Continuing after Tokens are Issued

A request MAY be continued even after access tokens have been issued, so long as the handle is valid. The AS MAY respond to such a continuation request with new access tokens as described in Section

3.2 based on the client's original request. The AS SHOULD revoke existing access tokens. If the AS determines that the client can make a further continuation request in the future, the AS MUST include a new "continue" response element (Section 3.1). The returned handle value MUST NOT be the same as that used to make the continuation request, and the continuation URI MAY remain the same. If the AS does not return a new "continue" response element, the client MUST NOT make an additional continuation request. If a client does so, the AS MUST return an error.

6. Token Management

If an access token response includes the "manage" parameter as described in Section 3.2.1, the client MAY call this URL to manage the access token with any of the actions defined in the following sections. Other actions are undefined by this specification.

The access token being managed acts as the access element for its own management API. The client MUST present proof of an appropriate key along with the access token.

If the token is sender-constrained (i.e., not a bearer token), it MUST be sent with the appropriate binding for the access token (Section 7).

If the token is a bearer token, the client MUST present proof of the same key identified in the initial request (Section 2.3) as described in Section 8.

The AS MUST validate the proof and assure that it is associated with either the token itself or the client the token was issued to, as appropriate for the token's presentation type.

6.1. Rotating the Access Token

The client makes an HTTP POST to the token management URI, sending the access token in the appropriate header and signing the request with the appropriate key.

```
POST /token/PRY5NM33OM4TB8N6BW7OZB8CDFONP219RP1L HTTP/1.1
Host: server.example.com
Authorization: GNAP OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0
Detached-JWS: eyj0....
```

The AS validates that the token presented is associated with the management URL, that the AS issued the token to the given client, and that the presented key is appropriate to the token. The access token MAY be expired, and in such cases the AS SHOULD honor the rotation request to the token management URL. The AS MAY store

different lifetimes for the use of the token in rotation vs. its use at an RS.

If the token is validated and the key is appropriate for the request, the AS will invalidate the current access token associated with this URL, if possible, and return a new access token response as described in Section 3.2.1. The value of the access token MUST NOT be the same as the current value of the access token used to access the management API. The response MAY include an updated access token management URL as well, and if so, the client MUST use this new URL to manage the new access token.

```
{
    "access_token": {
        "value": "FP6A8H6HY37MH13CK76LBZ6Y1UADG6VEUPEER5H2",
        "proof": "bearer",
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7O
        "resources": [
            {
                "type": "photo-api",
                "actions": [
                    "read",
                    "write",
                    "dolphin"
                ],
                "locations": [
                    "https://server.example.net/",
                    "https://resource.local/other"
                ],
                "datatypes": [
                    "metadata",
                    "images"
                ]
            },
            "read", "dolphin-metadata"
        ]
    }
}
```

6.2. Revoking the Access Token

The client makes an HTTP DELETE request to the token management URI, signing the request with its key.

```
DELETE /token/PRY5NM33OM4TB8N6BW7OZB8CDFONP219RP1L HTTP/1.1
Host: server.example.com
Authorization: GNAP OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0
Detached-JWS: eyj0....
```

If the token was issued to the client identified by the key, the AS will invalidate the current access token associated with this URL, if possible, and return an HTTP 204 response code.

```
204 No Content
```

7. Using Access Tokens

The method the RC uses to send an access token to the RS depends on the value of the "proof" parameter in the access token response (Section 3.2.1).

If this value is "bearer", the access token is sent using the HTTP Header method defined in [RFC6750].

```
Authorization: Bearer OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0
```

If the "proof" value is any other string, the access token is sent using the HTTP authorization scheme "GNAP" along with a key proof as described in Section 8 for the key bound to the access token. For example, a "jwsd"-bound access token is sent as follows:

```
Authorization: GNAP OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0
Detached-JWS: eyj0....
```

[[Editor's note: I don't actually like the idea of using only one header type for differently-bound access tokens, but instead these values should somehow reflect the key binding types. Maybe there can be multiple fields after the "GNAP" keyword using structured headers? Or a set of derived headers like GNAP-mtls? This might also be better as a separate specification, like OAuth 2.]]

8. Binding Keys

Any keys presented by the RC to the AS or RS MUST be validated as part of the request in which they are presented. The type of binding used is indicated by the proof parameter of the key section in the initial request Section 2.3. Values defined by this specification are as follows:

jwsd A detached JWS signature header

jws Attached JWS payload

mtls Mutual TLS certificate verification

dpop OAuth Demonstration of Proof-of-Possession key proof header

httpsig HTTP Signing signature header

oauthpop
: OAuth PoP key proof authentication header

Additional values can be defined by a registry TBD (Section 12).

The keys presented by the RC in the Section 2 MUST be proved in all continuation requests Section 5 and token management requests Section 6. The AS MUST validate all keys presented by the RC (Section 2.3) or referenced in an ongoing transaction at each call.

8.1. Detached JWS

This method is indicated by jwsd in the proof field. To sign a request, the RC takes the serialized body of the request and signs it using detached JWS [RFC7797]. The header of the JWS MUST contain the kid field of the key bound to this RC for this request. The JWS header MUST contain an alg field appropriate for the key identified by kid and MUST NOT be none.

The RC presents the signature in the Detached-JWS HTTP Header field. [Editor's Note: this is a custom header field, do we need this?]

```
POST /tx HTTP/1.1
Host: server.example.com
Content-Type: application/json
Detached-JWS: eyJiNjQiOmZhbHNlLCJhbGciOiJSUzI1NiIsImtpZCI6Inh5ei0xIn0.
  .Y287HMtaY0EegEjoTd_04a4GC6qV48GgVbGKOhHdJnDtD0VuUlVjLfwne8AuUY3U7e8
  9zUWwXLnAYK_BiS84M8EsrFvmv8yDLWzqveeIpcN5_ysveQnYt9Dqi32w6IOtAywkNUD
  ZeJEdc3z5s9Ei8qrYFN2fxcu28YS4e8e_cHTK57003WJu-wFn2TJUmAbHuqvUsyTb-nz
  YOKxuCKlqQItJF7E-cwSb_xULu-3f77BEU_vGbNYo5ZBa2B7UHO-kWNMSgbW2yeNNLbL
  C18Kv80GF22Y7SbZt0e2TwnR2Aa2zksuUbntQ5c7a1-gxtnXzuIKa34OekrnyqE1hmVW
  peQ

{
    "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
    },
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "key": {
        "proof": "jwsd",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
                    "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_JtffXyaSx8
xYJCNaOKNJn_Oz0YhdHbXTeWO5AoyspDWJbN5w_7bdWDxgpD-y6jnD1u9YhBOCWObNPF
vpkTM8LC7SdXGRKx2k8Me2r_GssYlyRpqvpBlY5-ejCywKRBfctRcnhTTGNztbbDBUyD
SWmFMVCHe5mXT4cL0BwrZC6S-uu-LAx06aKwQOPwYOGOslK8WPm1yGdkaA1uF_FpS6LS
63WYPHi_Ap2B7_8Wbw4ttzbMS_doJvuDagW8A1Ip3fXFAHtRAcKw7rdI4_Xln66hJxFe
kpdfWdiPQddQ6Y1cK2U3obvUg7w"
        }
    }
}
```

When the AS receives the Detached-JWS header, it MUST parse its contents as a detached JWS object. The HTTP Body is used as the payload for purposes of validating the JWS, with no transformations.

[[Editor's note: this is a potentially fragile signature mechanism. It doesn't protect the method or URL of the request in the signature, but it's simple to calculate and useful for body-driven requests, like the client to the AS. We might want to remove this in favor of general-purpose HTTP signing.]]

8.2. Attached JWS

This method is indicated by jws in the proof field. To sign a request, the RC takes the serialized body of the request JSON and signs it using JWS [RFC7515]. The header of the JWS MUST contain the kid field of the key bound to this RC during this request. The JWS header MUST contain an alg field appropriate for the key identified by kid and MUST NOT be none.

The RC presents the JWS as the body of the request along with a content type of application/jose. The AS MUST extract the payload of the JWS and treat it as the request body for further processing.

```
POST /transaction HTTP/1.1
Host: server.example.com
Content-Type: application/jose
```


eyJiNjQiOmZhbHNlLCJhbGciOiJSUzI1NiIsImtpZCI6Inh5ei0xIn0.ewogICAgIm
NsaWVudCI6IHsKICAgICAgICAibmFtZSI6ICJNeSBDbGllbnQgRGlzcGxheSBOYW1l
IiwKICAgICAgICAidXJpIjogImh0dHBzOi8vZXhhbXBsZS5uZXQvY2xpZW50IgogIC
AgfSwKICAgICJyZXNvdXJjZXMiOiBbCiAgICAgICAgImRvbHBoaW4tbWV0YWRhdGEi
CiAgICBdLAogICAgImludGVyYWN0IjogewogICAgICAgICJyZWRpcmVjdCI6IHRydW
UsCiAgICAgICAgImNhbGxiYWNrIjogewogICAgCQkidXJpIjogImh0dHBzOi8vY2xp
ZW50LmZvbyIsCiAgICAJCSJub25jZSI6ICJWSkxPNkE0Q0FZTEJYSFRSMEtSTyIKIC
AgIAl9CiAgICB9LAogICAgImtleXMiOiB7CgkJInByb29mIjogImp3c2QiLAogICAg
ICAgICJqd2tzIjogewogICAgICAgICAgICAia2V5cyI6IFsKICAgICAgICAgICAgIC
AgIHsKICAgICAgICAgICAgICAgICAgICAia3R5IjogIlJTQSIsCiAgICAgICAgICAg
ICAgICAgICAgImUiOiAiQVFBQiIsCiAgICAgICAgICAgICAgICAgICAgImtpZCI6IC
J4eXotMSIsCiAgICAgICAgICAgICAgICAgICAgImFsZyI6ICJSUzI1NiIsCiAgICAg
ICAgICAgICAgICAgICAgIm4iOiAia09CNXJSNEp2MEdNZUxhWTZfSXRfcjNPUndkZj
hjaV9KdGZmWHlhU3g4eFlKQ0NOYU9LTkpuX096MFloZEhiWFRlV081QW95c3BEV0pi
TjV3XzdiZFdEeGdwRC15NmpuRDF1OVloQk9DV09iTlBGdnBrVE04TEM3U2RYR1JLeD
JrOE1lMnJfR3NzWWx5UnBxdnBCbFk1LWVqQ3l3S1JCZmN0UmNuaFRUR056dGJiREJV
eURTV21GTVZDSGU1bVhUNGNMMEJ3clpDNlMtdXUtTEF4MDZhS3dRT1B3WU9HT3NsSz
hXUG0xeUdka2FBMXVGX0ZwUzZMUzYzV1lQSGlfQXAyQjdfOFdidzR0dHpiTVNfZG9K
dnVEYWdXOEExSXAzZlhGQUh0UkFjS3c3cmRJNF9YbG42NmhKeEZla3BkZldkaVBRZG
RRNlkxY0syVTNvYnZVZzd3IgogICAgICAgICAgICAgICAgfQogICAgICAgICAgICBd
CiAgICAgICAgfQogICAgfQp9.Y287HMtaY0EegEjoTd_04a4GC6qV48GgVbGKOhHdJ
nDtD0VuUlVjLfwne8AuUY3U7e89zUWwXLnAYK_BiS84M8EsrFvmv8yDLWzqveeIpcN
5_ysveQnYt9Dqi32w6IOtAywkNUDZeJEdc3z5s9Ei8qrYFN2fxcu28YS4e8e_cHTK5
7003WJu-wFn2TJUmAbHuqvUsyTb-nzYOKxuCKlqQItJF7E-cwSb_xULu-3f77BEU_v
GbNYo5ZBa2B7UHO-kWNMSgbW2yeNNLbLC18Kv80GF22Y7SbZt0e2TwnR2Aa2zksuUb
ntQ5c7a1-gxtnXzuIKa34OekrnyqE1hmVWpeQ

[[Editor's note: A downside to this method is that it requires the content type to be something other than application/json, and it doesn't work against an RS without additional profiling since it requires things to be sent in the body. Additionally it is potentially fragile like a detached JWS since a multi-tier system could parse the payload and pass the parsed payload downstream with potential transformations. Furthermore, it doesn't protect the method or URL of the request in the signature. We might want to remove this in favor of general-purpose HTTP signing.]]

8.3. Mutual TLS

This method is indicated by mtls in the proof field. The RC presents its client certificate during TLS negotiation with the server (either AS or RS). The AS or RS takes the thumbprint of the client certificate presented during mutual TLS negotiation and compares

that thumbprint to the thumbprint presented by the RC application as described in [RFC8705] section 3.

```
POST /transaction HTTP/1.1
Host: server.example.com
Content-Type: application/json
SSL_CLIENT_CERT: MIIEHDCCAwSgAwIBAgIBATANBgkqhkiG9w0BAQsFADCBmjE3MDUGA1U
 cG9rZSBFbmdpbmVlcmluZyBSb290IENlcnRpZmljYXRlIEF1dGhvcml0eTELMAkG
 A1UECAwCTUExCzAJBgNVBAYTAlVTMRkwFwYJKoZIhvcNAQkBFgpjYUBic3BrLmlv
 MRwwGgYDVQQKDBNCZXNwb2tlIEVuZ2luZWVyaW5nMQwwCgYDVQQLDANNVEkwHhcN
 MTkwNDEwMjE0MDI5WhcNMjQwNDA4MjE0MDI5WjB8MRIwEAYDVQQDDAlsb2NhbGhv
 c3QxCzAJBgNVBAgMAk1BMQswCQYDVQQGEwJVUzEgMB4GCSqGSIb3DQEJARYRdGxz
 Y2xpZW50QGJzcGsuaW8xHDAaBgNVBAoME0Jlc3Bva2UgRW5naW5lZXJpbmcxDDAK
 BgNVBAsMA01USTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMmaXQHb
 s/wc1RpsQ6Orzf6rN+q2ijaZbQxD8oi+XaaN0P/gnE13JqQduvdq77OmJ4bQLokq
 sd0BexnI07Njsl8nkDDYpe8rNve5TjyUDCfbwgS7U1CluYenXmNQbaYNDOmCdHww
 UjV4kKREg6DGAx22Oq7+VHPTeeFgyw4kQgWRSfDENWY3KUXJlb/vKR6lQ+aOJytk
 vj8kVZQtWupPbvwoJe0na/ISNAOhL74w20DWWoDKoNltXsEtflNljVoi5nqsmZQc
 jfjt6LO0T7O1OX3Cwu2xWx8KZ3n/2ocuRqKEJHqUGfeDtuQNt6Jz79v/OTr8puLW
 aD+uyk6NbtGjoQsCAwEAAaOBiTCBhjAJBgNVHRMEAjAAMAsGA1UdDwQEAwIF4DBs
 BgNVHREEZTBjgglsb2NhbGhvc3SCD3Rsc2NsaWVudC5sb2NhbIcEwKgBBIERdGxz
 Y2xpZW50QGJzcGsuaW+GF2h0dHA6Ly90bHNjbGllbnQubG9jYWwvhhNzc2g6dGxz
 Y2xpZW50LmxvY2FsMA0GCSqGSIb3DQEBCwUAA4IBAQCKKv8WlLrT4Z5NazaUrYtl
 TF+2v0tvZBQ7qzJQjlOqAcvxry/d2zyhiRCRS/v318YCJBEv4Iq2W3I3JMMyAYEe
 2573HzT7rH3xQP12yZyRQnetdiVM1Z1KaXwfrPDLs72hUeELtxIcfZ0M085jLboX
 hufHI6kqm3NCyCCTihe2ck5RmCc5l2KBO/vAHF0ihhFOOOby1v6qbPHQcxAU6rEb
 907/p6BW/LV1NCgYB1QtFSfGxowqb9FRIMD2kvMSmO0EMxgwZ6k6spa+jk0IsI3k
 lwLW9b+Tfn/daUbIDctxeJneq2anQyU2znBgQl6KILDSF4eaOqlBut/KNZHHazJh

{
    "client": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
    },
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "key": {
        "proof": "mtls",
        "cert": "MIIEHDCCAwSgAwIBAgIBATANBgkqhkiG9w0BAQsFADCBmjE3
MDUGA1UEAwwuQmVzcG9rZSBFbmdpbmVlcmluZyBSb290IENlcnRpZmljYXRlIEF1d
Ghvcml0eTELMAkGA1UECAwCTUExCzAJBgNVBAYTAlVTMRkwFwYJKoZIhvcNAQkBFg
pjYUBic3BrLmlvMRwwGgYDVQQKDBNCZXNwb2tlIEVuZ2luZWVyaW5nMQwwCgYDVQQ
LDANNVEkwHhcNMTkwNDEwMjE0MDI5WhcNMjQwNDA4MjE0MDI5WjB8MRIwEAYDVQQD
```

```
DAlsb2NhbGhvc3QxCzAJBgNVBAgMAk1BMQswCQYDVQQGEwJVUzEgMB4GCSqGSIb3D
QEJARYRdGxzY2xpZW50QGJzcGsuaW8xHDAaBgNVBAoME0Jlc3Bva2UgRW5naW5lZX
JpbmcxDDAKBgNVBAsMA01USTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggE
BAMmaXQHbs/wc1RpsQ6Orzf6rN+q2ijaZbQxD8oi+XaaN0P/gnE13JqQduvdq77Om
J4bQLokqsd0BexnI07Njsl8nkDDYpe8rNve5TjyUDCfbwgS7U1CluYenXmNQbaYND
OmCdHwwUjV4kKREg6DGAx22Oq7+VHPTeeFgyw4kQgWRSfDENWY3KUXJlb/vKR6lQ+
aOJytkvj8kVZQtWupPbvwoJe0na/ISNAOhL74w20DWWoDKoNltXsEtflNljVoi5nq
smZQcjfjt6LO0T7O1OX3Cwu2xWx8KZ3n/2ocuRqKEJHqUGfeDtuQNt6Jz79v/OTr8
puLWaD+uyk6NbtGjoQsCAwEAAaOBiTCBhjAJBgNVHRMEAjAAMAsGA1UdDwQEAwIF4
DBsBgNVHREEZTBjgglsb2NhbGhvc3SCD3Rsc2NsaWVudC5sb2NhbIcEwKgBBIERdG
xzY2xpZW50QGJzcGsuaW+GF2h0dHA6Ly90bHNjbGllbnQubG9jYWwvhhNzc2g6dGx
zY2xpZW50LmxvY2FsMA0GCSqGSIb3DQEBCwUAA4IBAQCKKv8WlLrT4Z5NazaUrYtl
TF+2v0tvZBQ7qzJQjlOqAcvxry/d2zyhiRCRS/v318YCJBEv4Iq2W3I3JMMyAYEe2
573HzT7rH3xQP12yZyRQnetdiVM1Z1KaXwfrPDLs72hUeELtxIcfZ0M085jLboXhu
fHI6kqm3NCyCCTihe2ck5RmCc5l2KBO/vAHF0ihhFOOOby1v6qbPHQcxAU6rEb907
/p6BW/LV1NCgYB1QtFSfGxowqb9FRIMD2kvMSmO0EMxgwZ6k6spa+jk0IsI3klwLW
9b+Tfn/daUbIDctxeJneq2anQyU2znBgQl6KILDSF4eaOqlBut/KNZHHazJh"
    }
}
```

8.4. DPoP

This method is indicated by dpop in the proof field. The RC creates a Demonstration of Proof-of-Possession signature header as described in [I-D.ietf-oauth-dpop] section 2.

```
POST /transaction HTTP/1.1
Host: server.example.com
Content-Type: application/json
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IlJTMjU2IiwiandrIjp7Imt0eSI6Il
JTQSIsImUiOiJBUUFCIiwia2lkIjoieHl6LWNsaWVudCIsImFsZyI6IlJTMjU2Iiwibi
I6Inp3Q1RfM2J4LWdsYmJIcmhlWXBZcFJXaVk5SS1uRWFNUnBablJySWpDczZiX2VteV
RrQmtEREVqU3lzaTM4T0M3M2hqMS1XZ3hjUGRLTkdaeUlvSDNRWmVuMU1LeXloUXBMSk
cxLW9MTkxxbTdwWFh0ZFl6U2RDOU8zLW9peXk4eWtPNFlVeU5aclJSZlBjaWhkUUNiT1
9PQzhRdWdtZzlyZ05ET1NxcHBkYU5lYXMxb3Y5UHhZdnhxcnoxLThIYTdna0QwMFlFQ1
hIYUIwNXVNYVVhZEhxLU9fV0l2WVhpY2c2STVqNlM0NFZOVTY1VkJ3dS1BbHluVHhRZE
1BV1AzYll4VlZ5NnAzLTdlVEpva3ZqWVRGcWdEVkRaOGxVWGJyNXlDVG5SaG5oSmd2Zj
NWakRfbWFsTmU4LXRPcUs1T1NEbEhUeTZnRDlOcWRHQ20tUG0zUSJ9fQ.eyJodHRwX21
ldGhvZCI6IlBPU1QiLCJodHRwX3VyaSI6Imh0dHA6XC9cL2hvc3QuZG9ja2VyLmludGV
ybmFsOjk4MzRcL2FwaVwvYXNcL3RyYW5zYWN0aW9uIiwiaWF0IjoxNTcyNjQyNjEzLCJ
qdGkiOiJIam9IcmpnbTJ5QjR4N2pBNXl5RyJ9.aUhftvfw2NoW3M7durkopReTvONng1
fOzbWjAlKNSLL0qIwDgfG39XUyNvwQ23OBIwe6IuvTQ2UBBPklPAfJhDTKd8KHEAfidN
B-LzUOzhDetLg30yLFzIpcEBMLCjb0TEsmXadvxuNkEzFRL-Q-QCg0AXSF1h57eAqZV8
SYF4CQK9OUV6fIWwxLDd3cVTx83MgyCNnvFlG_HDyim1Xx-rxV4ePd1vgDeRubFb6QWj
iKEO7vj1APv32dsux67gZYiUpjm0wEZprjlG0a07R984KLeK1XPjXgViEwEdlirUmpVy
T9tyEYqGrTfm5uautELgMls9sgSyE929woZ59elg

{
    "client": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
    },
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "key": {
        "proof": "dpop",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
                    "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_JtffXyaSx8xYJ
CCNaOKNJn_Oz0YhdHbXTeWO5AoyspDWJbN5w_7bdWDxgpD-y6jnD1u9YhBOCWObNPFvpkTM
8LC7SdXGRKx2k8Me2r_GssYlyRpqvpBlY5-ejCywKRBfctRcnhTTGNztbbDBUyDSWmFMVCH
e5mXT4cL0BwrZC6S-uu-LAx06aKwQOPwYOGOslK8WPm1yGdkaA1uF_FpS6LS63WYPHi_Ap2
B7_8Wbw4ttzbMS_doJvuDagW8A1Ip3fXFAHtRAcKw7rdI4_Xln66hJxFekpdfWdiPQddQ6Y
```

```
1cK2U3obvUg7w"
        }
    }
}
```

[[Editor's note: this method requires duplication of the key in the header and the request body, which is redundant and potentially awkward. The signature also doesn't protect the body of the request.]]

8.5. HTTP Signing

This method is indicated by httpsig in the proof field. The RC creates an HTTP Signature header as described in [I-D.ietf-httpbis-message-signatures] section 4. The RC MUST calculate and present the Digest header as defined in [RFC3230].

```
POST /transaction HTTP/1.1
Host: server.example.com
Content-Type: application/json
Content-Length: 716
Signature: keyId="xyz-client", algorithm="rsa-sha256",
 headers="(request-target) digest content-length",
 signature="TkehmgK7GD/z4jGkmcHS67cjVRgm3zVQNlNrrXW32Wv7d
u0VNEIVI/dMhe0WlHC93NP3ms91i2WOW5r5B6qow6TNx/82/6W84p5jqF
YuYfTkKYZ69GbfqXkYV9gaT++dl5kvZQjVk+KZT1dzpAzv8hdk9nO87Xi
rj7qe2mdAGE1LLc3YvXwNxuCQh82sa5rXHqtNT1077fiDvSVYeced0UEm
rWwErVgr7sijtbTohC4FJLuJ0nG/KJUcIG/FTchW9rd6dHoBnY43+3Dzj
CIthXpdH5u4VX3TBe6GJDO6Mkzc6vB+67OWzPwhYTplUiFFV6UZCsDEeu
Sa/Ue1yLEAMg==""]}
Digest: SHA=oZz2O3kg5SEFAhmr0xEBbc4jEfo=

{
    "client": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
    },
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "push",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "key": {
        "proof": "httpsig",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
                    "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_J
tffXyaSx8xYJCCNaOKNJn_Oz0YhdHbXTeWO5AoyspDWJbN5w_7bdWDxgpD-
y6jnD1u9YhBOCWObNPFvpkTM8LC7SdXGRKx2k8Me2r_GssYlyRpqvpBlY5-
ejCywKRBfctRcnhTTGNztbbDBUyDSWmFMVCHe5mXT4cL0BwrZC6S-uu-LAx
06aKwQOPwYOGOslK8WPm1yGdkaA1uF_FpS6LS63WYPHi_Ap2B7_8Wbw4ttz
bMS_doJvuDagW8A1Ip3fXFAHtRAcKw7rdI4_Xln66hJxFekpdfWdiPQddQ6
Y1cK2U3obvUg7w"
        }
    }
}
```

When used to present an access token as in Section 7, the Authorization header MUST be included in the signature.

8.6. OAuth PoP

This method is indicated by oauthpop in the proof field. The RC creates an HTTP Authorization PoP header as described in [I-D.ietf-oauth-signed-http-request] section 4, with the following additional requirements:

- The at (access token) field MUST be [note: this is in contrast to the requirements in the existing spec] unless this method is being used in conjunction with an access token as in Section 7.

- The b (body hash) field MUST be calculated and supplied

```
POST /transaction HTTP/1.1
Host: server.example.com
Content-Type: application/json
PoP: eyJhbGciOiJSUzI1NiIsImp3ayI6eyJrdHkiOiJSU0EiLCJlIjoi
QVFBQiIsImtpZCI6Inh5ei1jbGllbnQiLCJhbGciOiJSUzI1NiIsIm4iO
iJ6d0NUXzNieC1nbGJiSHJoZVlwWXBSV2lZOUktbkVhTVJwWm5ScklqQ3
M2Yl9lbXlUa0JrRERFalN5c2kzOE9DNzNoajEtV2d4Y1BkS05HWnlJb0g
zUVplbjFNS3l5aFFwTEpHMS1vTE5McW03cFhYdGRZelNkQzlPMy1vaXl5
OHlrTzRZVXlOWnJSUmZQY2loZFFDYk9fT0M4UXVnbWc5cmdORE9TcXBwZ
GFOZWFzMW92OVB4WXZ4cXJ6MS04SGE3Z2tEMDBZRUNYSGFCMDV1TWFVYW
RIcS1PX1dJdllYaWNnNkk1ajZTNDRWTlU2NVZCd3UtQWx5blR4UWRNQVd
QM2JZeFZWeTZwMy03ZVRKb2t2allURnFnRFZEWjhsVVhicjV5Q1RuUmhu
aEpndmYzVmpEX21hbE5lOC10T3FLNU9TRGxIVHk2Z0Q5TnFkR0NtLVBtM
1EifX0.eyJwIjoiXC9hcGlcL2FzXC90cmFuc2FjdGlvbiIsImIiOiJxa0
lPYkdOeERhZVBTZnc3NnFjamtqSXNFRmxDb3g5bTU5NFM0M0RkU0xBIiw
idSI6Imhvc3QuZG9ja2VyLmludGVybmFsIiwiaCI6W1siQWNjZXB0Iiwi
Q29udGVudC1UeXBlIiwiQ29udGVudC1MZW5ndGgiXSwiVjQ2OUhFWGx6S
k9kQTZmQU5oMmpKdFhTd3pjSGRqMUloOGk5M0h3bEVHYyJdLCJtIjoiUE
9TVCIsInRzIjoxNTcyNjQyNjEwfQ.xyQ47qy8bu4fyK1T3Ru1Sway8wp6
5rfAKnTQQU92AUUU07I2iKoBL2tipBcNCC5zLH5j_WUyjlN15oi_lLHym
fPdzihtt8_Jibjfjib5J15UlifakjQ0rHX04tPal9PvcjwnyZHFcKn-So
Y3wsARn-gGwxpzbsPhiKQP70d2eG0CYQMA6rTLslT7GgdQheelhVFW29i
27NcvqtkJmiAG6Swrq4uUgCY3zRotROkJ13qo86t2DXklV-eES4-2dCxf
cWFkzBAr6oC4Qp7HnY_5UT6IWkRJt3efwYprWcYouOVjtRan3kEtWkaWr
G0J4bPVnTI5St9hJYvvh7FE8JirIg

{
    "client": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
    },
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "key": {
        "proof": "oauthpop",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
```

```
                    "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_J
tffXyaSx8xYJCCNaOKNJn_Oz0YhdHbXTeWO5AoyspDWJbN5w_7bdWDxgpD-
y6jnD1u9YhBOCWObNPFvpkTM8LC7SdXGRKx2k8Me2r_GssYlyRpqvpBlY5-
ejCywKRBfctRcnhTTGNztbbDBUyDSWmFMVCHe5mXT4cL0BwrZC6S-uu-LAx
06aKwQOPwYOGOslK8WPm1yGdkaA1uF_FpS6LS63WYPHi_Ap2B7_8Wbw4ttz
bMS_doJvuDagW8A1Ip3fXFAHtRAcKw7rdI4_Xln66hJxFekpdfWdiPQddQ6
Y1cK2U3obvUg7w"
        }
    }
}
```

9. Discovery

By design, the protocol minimizes the need for any pre-flight discovery. To begin a request, the RC only needs to know the endpoint of the AS and which keys it will use to sign the request. Everything else can be negotiated dynamically in the course of the protocol.

However, the AS can have limits on its allowed functionality. If the RC wants to optimize its calls to the AS before making a request, it MAY send an HTTP OPTIONS request to the transaction endpoint to retrieve the server's discovery information. The AS MUST respond with a JSON document containing the following information:

grant_request_endpoint REQUIRED. The full URL of the AS's grant request endpoint. This MUST match the URL the RC used to make the discovery request.

capabilities OPTIONAL. A list of the AS's capabilities. The values of this result MAY be used by the RC in the capabilities section (Section 2.7) of the request.

interaction_methods OPTIONAL. A list of the AS's interaction methods. The values of this list correspond to the possible fields in the interaction section (Section 2.5) of the request.

key_proofs OPTIONAL. A list of the AS's supported key proofing mechanisms. The values of this list correspond to possible values of the proof field of the key section (Section 2.3) of the request.

sub_ids OPTIONAL. A list of the AS's supported identifiers. The values of this list correspond to possible values of the subject identifier section (Section 2.2) of the request.

assertions OPTIONAL. A list of the AS's supported assertion formats. The values of this list correspond to possible values of the subject assertion section (Section 2.2) of the request.

The information returned from this method is for optimization purposes only. The AS MAY deny any request, or any portion of a request, even if it lists a capability as supported. For example, a given client can be registered with the mtls key proofing mechanism, but the AS also returns other proofing methods, then the AS will deny a request from that client using a different proofing mechanism.

10. Resource Servers

In some deployments, a resource server will need to be able to call the AS for a number of functions.

[[Editor's note: This section is for discussion of possible advanced functionality. It seems like it should be a separate document or set of documents, and it's not even close to being well-baked. This also adds additional endpoints to the AS, as this is separate from the token request process, and therefore would require RS-facing discovery or configuration information to make it work. Also-also, it does presume the RS can sign requests in the same way that a client does, but hopefully we can be more consistent with this than RFC7662 was able to do.]]

10.1. Introspecting a Token

When the RS receives an access token, it can call the introspection endpoint at the AS to get token information. [[Editor's note: this isn't super different from the token management URIs, but the RS has no way to get that URI, and it's bound to different keys.]]

The RS signs the request with its own key and sends the access token as the body of the request.

```
POST /introspect HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

{
    "access_token": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
}
```

The AS responds with a data structure describing the token's current state and any information the RS would need to validate the token's presentation, such as its intended proofing mechanism and key material.

```
Content-type: application/json

{
    "active": true,
    "resources": [
        "dolphin-metadata", "some other thing"
    ],
    "resources": [
        "dolphin-metadata", "some other thing"
    ],
    "proof": "httpsig",
    "key": {
        "proof": "jwsd",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
                    "n": "kOB5rR4Jv0GMeL...."
        }
    }
}
```

10.2. Deriving a downstream token

If the RS needs to derive a token from one presented to it, it can request one from the AS by making a token request as described in Section 2 and presenting the existing access token's value in the "existing_access_token" field.

The RS MUST identify itself with its own key and sign the request.

[[Editor's note: this is similar to but based on the access token and not the grant. The fact that the keys presented are not the ones used for the access token should indicate that it's a different party and a different kind of request.]]

```
POST /tx HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

{
    "resources": [
        {
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        },
        "dolphin-metadata"
    ],
    "key": "7C7C4AZ9KHRS6X63AJAO",
    "existing_access_token": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0"
}
```

The AS responds with a token as described in Section 3.

10.3. Registering a Resource Handle

If the RS needs to, it can post a set of resources as described in Section 2.1.1 to the AS's resource registration endpoint.

The RS MUST identify itself with its own key and sign the request.

```
POST /resource HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

{
    "resources": [
        {
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        },
        "dolphin-metadata"
    ],
    "key": "7C7C4AZ9KHRS6X63AJAO"

}
```

The AS responds with a handle appropriate to represent the resources list that the RS presented.

```
Content-type: application/json

{
    "resource_handle": "FWWIKYBQ6U56NL1"
}
```

The RS MAY make this handle available as part of a response to a client (Section 10.4) or as documentation to developers.

[[Editor's note: It's not an exact match here because the "resource_handle" returned now represents a collection of objects instead of a single one. Perhaps we should let this return a list of strings instead? Or use a different syntax than the resource request? Also, this borrows heavily from UMA 2's "distributed authorization" model and, like UMA, might be better suited to an extension than the core protocol.]]

10.4. Requesting a Resources With Insufficient Access

If the client calls an RS without an access token, or with an invalid access token, the RS MAY respond to the client with an authentication header indicating that GNAP. The address of the GNAP endpoint MUST be sent in the "as_uri" parameter. The RS MAY additionally return a resource reference that the client MAY use in its resource request (Section 2.1). This resource reference handle SHOULD be sufficient for at least the action the client was attempting to take at the RS. The RS MAY use the dynamic resource handle request (Section 10.3) to register a new resource handle, or use a handle that has been pre-configured to represent what the AS is protecting. The content of this handle is opaque to the RS and the client.

```
WWW-Authenticate: GNAP as_uri=http://server.example/transaction,resource
```

The client then makes a call to the "as_uri" as described in Section 2, with the value of "resource" as one of the members of a "resources" array Section 2.1.1. The client MAY request additional resources and other information, and MAY request multiple access tokens.

[[Editor's note: this borrows heavily from UMA 2's "distributed authorization" model and, like UMA, might be better suited to an extension than the core protocol.]]

11. Acknowledgements

The author would like to thank the feedback of the following individuals for their reviews, implementations, and contributions: Aaron Parecki, Annabelle Backman, Dick Hardt, Dmitri Zagidulin, Dmitry Barinov, Fabien Imbault, Francis Pouatcha, George Fletcher, Haardik Haardik, Hamid Massaoud, Jacky Yuan, Joseph Heenan, Kathleen Moriarty, Mike Jones, Mike Varley, Nat Sakimura, Takahiko Kawasaki, Takahiro Tsuchiya.

In particular, the author would like to thank Aaron Parecki and Mike Jones for insights into how to integrate identity and authentication systems into the core protocol, and to Dick Hardt for the use cases, diagrams, and insights provided in the XAuth proposal that have been incorporated here. The author would like to especially thank Mike Varley and the team at SecureKey for feedback and development of early versions of the XYZ protocol that fed into this standards work.

12. IANA Considerations

[[TBD: There are a lot of items in the document that are expandable through the use of value registries.]]

13. Security Considerations

[[TBD: There are a lot of security considerations to add.]]

All requests have to be over TLS or equivalent as per [BCP195]. Many handles act as shared secrets, though they can be combined with a requirement to provide proof of a key as well.

14. Privacy Considerations

[[TBD: There are a lot of privacy considerations to add.]]

Handles are passed between parties and therefore should not contain any private data.

When user information is passed to the client, the AS needs to make sure that it has the permission to do so.

15. Normative References

[BCP195] "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", 2015, <http://www.rfc-editor.org/info/bcp195>.

[I-D.ietf-httpbis-message-signatures] Backman, A., Richer, J., and M. Sporny, "Signing HTTP Messages", Work in Progress, Internet-Draft, draft-ietf-httpbis-message-signatures-00, 10 April 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-message-signatures-00.txt>.

[I-D.ietf-oauth-dpop] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer (DPoP)", Work in Progress, Internet-Draft, draft-ietf-oauth-dpop-01, 1 May 2020, <http://www.ietf.org/internet-drafts/draft-ietf-oauth-dpop-01.txt>.

[I-D.ietf-oauth-signed-http-request] Richer, J., Bradley, J., and H. Tschofenig, "A Method for Signing HTTP Requests for OAuth", Work in Progress, Internet-Draft, draft-ietf-oauth-signed-http-request-03, 8 August 2016, <http://www.ietf.org/internet-drafts/draft-ietf-oauth-signed-http-request-03.txt>.

[I-D.ietf-secevent-subject-identifiers] Backman, A. and M. Scurtescu, "Subject Identifiers for Security Event Tokens", Work in Progress, Internet-Draft, draft-ietf-secevent-subject-identifiers-05, 24 July 2019,

<http://www.ietf.org/internet-drafts/draft-ietf-secevent-subject-identifiers-05.txt>.

[OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <https://openiD.net/specs/openiD-connect-core-1_0.html>.

[OIDC4IA] Lodderstedt, T. and D. Fett, "OpenID Connect for Identity Assurance 1.0", October 2019, <https://openid.net/specs/openid-connect-4-identity-assurance-1_0.html>.

[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/info/rfc2119>.

[RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <https://www.rfc-editor.org/info/rfc3230>.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7797] Jones, M., "JSON Web Signature (JWS) Unencoded Payload Option", RFC 7797, DOI 10.17487/RFC7797, February 2016, <https://www.rfc-editor.org/info/rfc7797>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI

10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/info/rfc8705>.

Appendix A. Document History

*-10

- Switched to xml2rfc v3 and markdown source.
- Updated based on Design Team feedback and reviews.
- Added acknowledgements list.
- Added sequence diagrams and explanations.
- Collapsed "short_redirect" into regular redirect request.
- Separated pass-by-reference into subsections.
- Collapsed "callback" and "pushback" into a single mode-switched method.
- Add OIDC Claims request object example.

*-09

- Major document refactoring based on request and response capabilities.
- Changed from "claims" language to "subject identifier" language.
- Added "pushback" interaction capability.
- Removed DIDCOMM interaction (better left to extensions).
- Excised "transaction" language in favor of "Grant" where appropriate.
- Added token management URLs.
- Added separate continuation URL to use continuation handle with.
- Added RS-focused functionality section.
- Added notion of extending a grant request based on a previous grant.
- Simplified returned handle structures.

*-08

- Added attached JWS signature method.

- Added discovery methods.

*-07

- Marked sections as being controlled by a future registry TBD.

*-06

- Added multiple resource requests and multiple access token response.

*-05

- Added "claims" request and response for identity support.

- Added "capabilities" request for inline discovery support.

*-04

- Added crypto agility for callback return hash.

- Changed "interaction_handle" to "interaction_ref".

*-03

- Removed "state" in favor of "nonce".

- Created signed return parameter for front channel return.

- Changed "client" section to "display" section, as well as associated handle.

- Changed "key" to "keys".

- Separated key proofing from key presentation.

- Separated interaction methods into booleans instead of "type" field.

*-02

- Minor editorial cleanups.

*-01

- Made JSON multimodal for handle requests.

- Major updates to normative language and references throughout document.

- Allowed interaction to split between how the user gets to the AS and how the user gets back.

* -00

- Initial submission.

Appendix B. Component Data Models

While different implementations of this protocol will have different realizations of all the components and artifacts enumerated here, the nature of the protocol implies some common structures and elements for certain components. This appendix seeks to enumerate those common elements.

TBD: Client has keys, allowed requested resources, identifier(s), allowed requested subjects, allowed

TBD: AS has "grant endpoint", interaction endpoints, store of trusted client keys, policies

TBD: Token has RO, user, client, resource list, RS list,

Appendix C. Example Protocol Flows

The protocol defined in this specification provides a number of features that can be combined to solve many different kinds of authentication scenarios. This section seeks to show examples of how the protocol would be applied for different situations.

Some longer fields, particularly cryptographic information, have been truncated for display purposes in these examples.

C.1. Redirect-Based User Interaction

In this scenario, the user is the RO and has access to a web browser, and the client can take front-channel callbacks on the same device as the user. This combination is analogous to the OAuth 2 Authorization Code grant type.

The client initiates the request to the AS. Here the client identifies itself using its public key.

```
POST /tx HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

{
    "resources": [
        {
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        }
    ],
    "key": {
        "proof": "jwsd",
        "jwk": {
            "kty": "RSA",
            "e": "AQAB",
            "kid": "xyz-1",
            "alg": "RS256",
            "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_JtffXyaSx8xY..."
        }
    },
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.example.net/return/123455",
            "nonce": "LKLTI25DK82FX4T4QFZC"
        }
    }
}
```

The AS processes the request and determines that the RO needs to interact. The AS returns the following response giving the client the information it needs to connect. The AS has also indicated to the client that it can use the given key handle to identify itself in future calls.

```
Content-type: application/json

{
    "interact": {
       "redirect": "https://server.example.com/interact/4CF492MLVMSW9MKM
       "callback": "MBDOFXG4Y5CVJCX821LH"
    }
    "continue": {
        "handle": "80UPRY5NM33OMUKMKSKU",
        "uri": "https://server.example.com/continue"
    },
    "key_handle": "7C7C4AZ9KHRS6X63AJAO"
}
```

The client saves the response and redirects the user to the interaction_url by sending the following HTTP message to the user's browser.

```
HTTP 302 Found
Location: https://server.example.com/interact/4CF492MLVMSW9MKMXKHQ
```

The user's browser fetches the AS's interaction URL. The user logs in, is identified as the RO for the resource being requested, and approves the request. Since the AS has a callback parameter, the AS generates the interaction reference, calculates the hash, and redirects the user back to the client with these additional values added as query parameters.

```
HTTP 302 Found
Location: https://client.example.net/return/123455
  ?hash=p28jsq0Y2KK3WS__a42tavNC64ldGTBroywsWxT4md_jZQ1R2HZT8BOWYHcLmObM
  &interact_ref=4IFWWIKYBC2PQ6U56NL1
```

The client receives this request from the user's browser. The client ensures that this is the same user that was sent out by validating session information and retrieves the stored pending request. The client uses the values in this to validate the hash parameter. The client then calls the continuation URL and presents the handle and interaction reference in the request body. The client signs the request as above.

```
POST /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...


{
    "handle": "80UPRY5NM33OMUKMKSKU",
    "interact_ref": "4IFWWIKYBC2PQ6U56NL1"
}
```

The AS retrieves the pending request based on the handle and issues a bearer access token and returns this to the client.

```
Content-type: application/json

{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "proof": "bearer",
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7O
        "resources": [{
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        }]
    },
    "continue": {
        "handle": "80UPRY5NM33OMUKMKSKU",
        "uri": "https://server.example.com/continue"
    }
}
```

C.2. Secondary Device Interaction

In this scenario, the user does not have access to a web browser on the device and must use a secondary device to interact with the AS. The client can display a user code or a printable QR code. The client prefers a short URL if one is available, with a maximum of

255 characters in length. The is not able to accept callbacks from the AS and needs to poll for updates while waiting for the user to authorize the request.

The client initiates the request to the AS.

```
POST /tx HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

{
    "resources": [
        "dolphin-metadata", "some other thing"
    ],
    "key": "7C7C4AZ9KHRS6X63AJAO",
    "interact": {
        "redirect": 255,
        "user_code": true
    }
}
```

The AS processes this and determines that the RO needs to interact. The AS supports both long and short redirect URIs for interaction, so it includes both. Since there is no "callback" the AS does not include a nonce, but does include a "wait" parameter on the continuation section because it expects the client to poll for results.

```
Content-type: application/json

{
    "interact": {
        "redirect": "https://srv.ex/MXKHQ",
        "user_code": {
            "code": "A1BC-3DFF",
            "url": "https://srv.ex/device"
        }
    },
    "continue": {
        "handle": "80UPRY5NM33OMUKMKSKU",
        "uri": "https://server.example.com/continue",
        "wait": 60
    }
}
```

The client saves the response and displays the user code visually on its screen along with the static device URL. The client also displays the short interaction URL as a QR code to be scanned.

If the user scans the code, they are taken to the interaction endpoint and the AS looks up the current pending request based on the incoming URL. If the user instead goes to the static page and enters the code manually, the AS looks up the current pending request based on the value of the user code. In both cases, the user logs in, is identified as the RO for the resource being requested, and approves the request. Once the request has been approved, the AS displays to the user a message to return to their device.

Meanwhile, the client periodically polls the AS every 60 seconds at the continuation URL.

```
POST /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...


{
    "handle": "80UPRY5NM33OMUKMKSKU"
}
```

The AS retrieves the pending request based on the handle and determines that it has not yet been authorized. The AS indicates to the client that no access token has yet been issued but it can continue to call after another 60 second timeout.

```
Content-type: application/json

{
    "continue": {
        "handle": "BI9QNW6V9W3XFJK4R02D",
        "uri": "https://server.example.com/continue",
        "wait": 60
    }
}
```

Note that the continuation handle has been rotated since it was used by the client to make this call. The client polls the continuation URL after a 60 second timeout using the new handle.

```
POST /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...


{
    "handle": "BI9QNW6V9W3XFJK4R02D"
}
```

The AS retrieves the pending request based on the handle and determines that it has been approved and it issues an access token.

```
Content-type: application/json

{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "proof": "bearer",
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7O
        "resources": [
            "dolphin-metadata", "some other thing"
        ]
    }
}
```

Appendix D. No User Involvement

In this scenario, the client is requesting access on its own behalf, with no user to interact with.

The client creates a request to the AS, identifying itself with its public key and using MTLS to make the request.

```
POST /tx HTTP/1.1
Host: server.example.com
Content-type: application/json

{
    "resources": [
        "backend service", "nightly-routine-3"
    ],
    "key": {
        "proof": "mtls",
        "cert#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
    }
}
```

The AS processes this and determines that the client can ask for the requested resources and issues an access token.

```
Content-type: application/json

{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "proof": "bearer",
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7O
        "resources": [
            "backend service", "nightly-routine-3"
        ]
    }
}
```

D.1. Asynchronous Authorization

In this scenario, the client is requesting on behalf of a specific RO, but has no way to interact with the user. The AS can asynchronously reach out to the RO for approval in this scenario.

The client starts the request at the AS by requesting a set of resources. The client also identifies a particular user.

```
POST /tx HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

{
    "resources": [
        {
            "type": "photo-api",
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        },
        "read", "dolphin-metadata",
        {
            "type": "financial-transaction",
            "actions": [
                "withdraw"
            ],
            "identifier": "account-14-32-32-3",
            "currency": "USD"
        },
        "some other thing"
    ],
    "key": "7C7C4AZ9KHRS6X63AJAO",
    "user": {
        "sub_ids": [ {
            "subject_type": "email",
            "email": "user@example.com"
        } ]
   }
}
```

The AS processes this and determines that the RO needs to interact. The AS determines that it can reach the identified user asynchronously and that the identified user does have the ability to approve this request. The AS indicates to the client that it can poll for continuation.

```
Content-type: application/json

{
    "continue": {
        "handle": "80UPRY5NM33OMUKMKSKU",
        "uri": "https://server.example.com/continue",
        "wait": 60
    }
}
```

The AS reaches out to the RO and prompts them for consent. In this example, the AS has an application that it can push notifications in to for the specified account.

Meanwhile, the client periodically polls the AS every 60 seconds at the continuation URL.

```
POST /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...


{
    "handle": "80UPRY5NM33OMUKMKSKU"
}
```

The AS retrieves the pending request based on the handle and determines that it has not yet been authorized. The AS indicates to the client that no access token has yet been issued but it can continue to call after another 60 second timeout.

```
Content-type: application/json

{
    "continue": {
        "handle": "BI9QNW6V9W3XFJK4R02D",
        "uri": "https://server.example.com/continue",
        "wait": 60
    }
}
```

Note that the continuation handle has been rotated since it was used by the client to make this call. The client polls the continuation URL after a 60 second timeout using the new handle.

```
POST /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...


{
    "handle": "BI9QNW6V9W3XFJK4R02D"
}
```

   The AS retrieves the pending request based on the handle and
   determines that it has been approved and it issues an access token.

```
Content-type: application/json

{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "proof": "bearer",
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7O
        "resources": [
            "dolphin-metadata", "some other thing"
        ]
    }
}
```

D.2. Applying OAuth 2 Scopes and Client IDs

   In this scenario, the client developer has a client_id and set of
   scope values from their OAuth 2 [RFC6749] system and wants to apply
   them to the new protocol. Traditionally, the OAuth 2 client
   developer would put their client_id and scope values as parameters
   into a redirect request to the authorization endpoint.

```
HTTP 302 Found
Location: https://server.example.com/authorize
  ?client_id=7C7C4AZ9KHRS6X63AJAO
  &scope=read%20write%20dolphin
  &redirect_uri=https://client.example.net/return
  &response_type=code
  &state=123455
```

   Now the developer wants to make an analogous request to the AS using
   the new protocol. To do so, the client makes an HTTP POST and places
   the OAuth 2 values in the appropriate places.

```
POST /tx HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

{
    "resources": [
        "read", "write", "dolphin"
    ],
    "key": "7C7C4AZ9KHRS6X63AJAO",
    "interact": {
        "redirect": true,
        "callback": {
            "uri": "https://client.example.net/return?state=123455",
            "nonce": "LKLTI25DK82FX4T4QFZC"
        }
    }
}
```

The client_id can be used to identify the client's keys that it uses for authentication, the scopes represent resources that the client is requesting, and the redirect_uri and state value are combined into a callback URI that can be unique per request. The client additionally creates a nonce to protect the callback, separate from the state parameter that it has added to its return URL.

From here, the protocol continues as above.

Author's Address

Justin Richer (editor)
Bespoke Engineering

Email: ietf@justin.richer.org
URI: https://bspk.io/