GNAP
Internet-Draft
Intended status: Standards Track
Expires: 21 May 2021

J. Richer, Ed.
Bespoke Engineering
A. Parecki
Okta
F. Imbault
acert.io

17 November 2020

Grant Negotiation and Authorization Protocol
draft-ietf-gnap-core-protocol-02

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 has been prepared by the GNAP working group design team of Kathleen Moriarty, Fabien Imbault, Dick Hardt, Mike Jones, and Justin Richer. This document is intended as a starting point for the working group and includes decision points for discussion and agreement. Many of the features in this proposed protocol can be accomplished in a number of ways. Where possible, the editor has included notes and discussion from the design team regarding the options as understood.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This protocol allows a piece of software, the resource client, to request delegated authorization to resource servers and to request direct information. This delegation is facilitated by an authorization server usually on behalf of a resource owner. The requesting party 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 GNAP allows for the negotiation of the grant process over time by multiple parties acting in distinct roles.

This protocol solves many of the same use cases as OAuth 2.0 [RFC6749], OpenID Connect [OIDC], and the family of protocols that have grown up around that ecosystem. However, GNAP is not an extension of OAuth 2.0 and is not intended to be directly compatible with OAuth 2.0. GNAP seeks to provide functionality and solve use cases that OAuth 2.0 cannot easily or cleanly address. Even so, GNAP and OAuth 2.0 will exist in parallel for many deployments, and considerations have been taken to facilitate the mapping and transition from legacy systems to GNAP. Some examples of these can be found in Appendix D.2.

1.1. Terminology

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.

1.2. Roles

The parties in GNAP perform actions under different roles. Roles are defined by the actions taken and the expectations leveraged on the role by the overall protocol.

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.

Resource Client (RC, aka "client") Requests tokens from the AS and uses tokens at the RS. An instance of the RC software is identified by its key, which can be known to the AS prior to the first request. The AS determines which policies apply to a given RC, including what it can request and on whose behalf.

Resource Server (RS, aka "API") Accepts tokens from the RC issued by the AS and serves delegated resources on behalf of the RO. There could be multiple RSs protected by the AS that the RC will call.

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

Requesting Party (RQ, aka "user") Operates and interacts with the RC.

The design of GNAP does not assume any one deployment architecture, but instead attempts to define roles that can be fulfilled in a number of different ways for different use cases. As long as a given role fulfills all of its obligations and behaviors as defined by the protocol, GNAP does not make additional requirements on its structure or setup.

Multiple roles can be fulfilled by the same party, and a given party can switch roles in different instances of the protocol. For example, the RO and RQ in many instances are the same person, where a user is authorizing the RC to act on their own behalf at the RS. In this case, one party fulfills both of the RO and RQ roles, but the roles themselves are still defined separately from each other to allow for other use cases where they are fulfilled by different parties.

For another example, in some complex scenarios, an RS receiving requests from one RC can act as an RC for a downstream secondary RS in order to fulfill the original request. In this case, one piece of software is both an RS and an RC from different perspectives, and it fulfills these roles separately as far as the overall protocol is concerned.

A single role need not be deployed as a monolithic service. For example, An RC could have components that are installed on the RQ's device as well as a back-end system that it communicates with. If both of these components participate in the delegation protocol, they are both considered part of the RC.

For another example, an AS could likewise be built out of many constituent components in a distributed architecture. The component that the RC calls directly could be different from the component that the the RO interacts with to drive consent, since API calls and user interaction have different security considerations in many environments. Furthermore, the AS could need to collect identity claims about the RO from one system that deals with user attributes while generating access tokens at another system that deals with security rights. From the perspective of GNAP, all of these are pieces of the AS and together fulfill the role of the AS as defined by the protocol.

[[See issue #29 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/29)]]

[[See issue #32 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/32)]]

1.3. Elements

In addition to the roles above, the protocol also involves several elements that are acted upon by the roles throughout the process.

Access Token A credential representing a set of access rights delegated to the RC. The access token is created by the AS, consumed and verified by the RS, and issued to and carried by the RC. The contents and format of the access token are opaque to the RC.

Grant The process by which the RC requests and is given delegated access to the RS by the AS through the authority of the RO.

Cryptographic Key A cryptographic element binding a request to a holder of the key. Access tokens and RC instances can be associated with specific keys.

Resource A protected API served by the RS and accessed by the RC. Access to this resource is delegated by the RO as part of the grant process.

Subject Information Information about the RO that is returned directly to the RC from the AS without the RC making a separate call to an RS. Access to this information is delegated by the RO as part of the grant process.

[[See issue #33 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/33)]]

1.4. Sequences

GNAP 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. For example, for an RC interested in only getting subject information directly, and not calling an RS, all steps involving the RS below do not apply.

In some circumstances, the information needed at a given stage is communicated out of band or is preconfigured 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.

```
        +------------+             +------------+
        | Requesting | ~ ~ ~ ~ ~ ~ |  Resource  |
        | Party (RQ) |             | Owner (RO) |
        +------------+             +------------+
            +                            +
            +                            +
           (A)                          (B)
            +                            +
            +                            +
        +--------+                       +       +------------+
        |Resource|--------------(1)------+------>|  Resource  |
        | Client |                       +       |   Server   |
        |  (RC)  |       +---------------+       |    (RS)    |
        |        |--(2)->| Authorization |       |            |
        |        |<-(3)--|     Server    |       |            |
        |        |       |      (AS)     |       |            |
        |        |--(4)->|               |       |            |
        |        |<-(5)--|               |       |            |
        |        |--------------(6)------------->|            |
        |        |       |               |<~(7)~~|            |
        |        |<-------------(8)------------->|            |
        |        |--(9)->|               |       |            |
        |        |<-(10)-|               |       |            |
        |        |--------------(11)------------>|            |
        |        |       |               |<~(12)~|            |
        |        |-(13)->|               |       |            |
        |        |       |               |       |            |
        +--------+       +---------------+       +------------+
```

```
    Legend
    + + + indicates a possible interaction with a human
    ----- indicates an interaction between protocol roles
    ~ ~ ~ indicates a potential equivalence or out-of-band communication between
roles
```

* (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. Note that for most situations, the RC already knows which AS to talk to and which kinds of access it needs.

* (2) The RC 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). Note that the RS could also examine the token directly, call an internal data store, execute a policy engine request, or any number of alternative methods for validating the token and its fitness for the request.

* (8) The RC to call the RS (Section 7) using the access token until the RS or RC determine that the token is no longer valid.

* (9) When the token no longer works, the RC fetches an updated access token (Section 6.1) based on the rights granted in (5).

* (10) The AS issues a new access token (Section 3.2) to the RC.

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

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

* (13) The RC disposes of the token (Section 6.2) once the RC has completed its access of the RS and no longer needs the token.

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

1.4.1. 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.

```
 +--------+                                  +--------+         +------+
 |   RC   |                                  |   AS   |         |  RO  |
 |        |                                  |        |         |  +   |
 |        |< (1) + Start Session + + + + + + + + + + + + + + + + |  RQ  |
 |        |                                  |        |         |(User)|
 |        |--(2)--- Request Access --------->|        |         |      |
 |        |                                  |        |         |      |
 |        |<-(3)-- Interaction Needed -------|        |         |      |
 |        |                                  |        |         |      |
 |        |+ (4) + Redirect for Interaction + + + + + + + + + > |      |
 |        |                                  |        |         |      |
 |        |                                  |        |<+ (5) +>|      |
 |        |                                  |        |  AuthN  |      |
 |        |                                  |        |         |      |
 |        |                                  |        |<+ (6) +>|      |
 |        |                                  |        |  AuthZ  |      |
 |        |                                  |        |         |      |
 |        |< (7) + Redirect for Continuation + + + + + + + + + + |      |
 |        |                                  |        |         +------+
 |        |--(8)--- Continue Request ------->|        |
 |        |                                  |        |
 |        |<-(9)----- Grant Access ----------|        |
 |        |                                  |        |
 +--------+                                  +--------+
```

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. Note that the RC needs to ensure that the parameters for the incoming request match those that it is expecting from the session created in (1). The RC also needs to be prepared for the RQ never being returned to the RC and handle time outs appropriately.

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.

1.4.2. User-code 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.

```
    +--------+                                  +--------+         +------+
    |   RC   |                                  |   AS   |         |  RO  |
    |        |--(1)--- Request Access --------->|        |         |  +   |
    |        |                                  |        |         |  RQ  |
    |        |<-(2)-- Interaction Needed -------|        |         |(User)|
    |        |                                  |        |         |      |
    |        |+ (3) + + Display User Code + + + + + + + + + + + + >|      |
    |        |                                  |        |         |      |
    |        |                                  |        |<+ (4) + |      |
    |        |                                  |        |Open URI |      |
    |        |                                  |        |         |      |
    |        |                                  |        |<+ (5) +>|      |
    |        |                                  |        |  AuthN  |      |
    |        |--(9)--- Continue Request (A) --->|        |         |      |
    |        |                                  |        |<+ (6) +>|      |
    |        |<-(10)- Not Yet Granted (Wait) ---|        |  Code   |      |
    |        |                                  |        |         |      |
    |        |                                  |        |<+ (7) +>|      |
    |        |                                  |        |  AuthZ  |      |
    |        |                                  |        |         |      |
    |        |                                  |        |<+ (8) +>|      |
    |        |                                  |        |Completed|      |
    |        |                                  |        |         |      |
    |        |--(11)-- Continue Request (B) --->|        |         +------+
    |        |                                  |        |
    |        |<-(12)----- Grant Access ---------|        |
    |        |                                  |        |
    +--------+                                  +--------+
```

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 RC's documentation, the AS documentation, or the RC software itself. Since it is assumed that the RO will interact with the AS through a secondary device, the RC does not provide a mechanism to launch the RO's browser at this URL.

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

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

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

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

9. 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.

10. 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 polled continuation requests. This response can include updated continuation information 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). Note that the AS may need to determine that the RO has not approved the request in a sufficient amount of time and return an appropriate error to the RC.

11. The RC continues to poll the AS (Section 5.2) with the new continuation information in (9).

12. 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.

1.4.3. 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.

```
    +--------+                                  +--------+         +------+
    |   RC   |                                  |   AS   |         |  RO  |
    |        |--(1)--- Request Access --------->|        |         |      |
    |        |                                  |        |         |      |
    |        |<-(2)-- Not Yet Granted (Wait) ---|        |         |      |
    |        |                                  |        |<+ (3) +>|      |
    |        |                                  |        |  AuthN  |      |
    |        |--(6)--- Continue Request (A) --->|        |         |      |
    |        |                                  |        |<+ (4) +>|      |
    |        |<-(7)-- Not Yet Granted (Wait) ---|        |  AuthZ  |      |
    |        |                                  |        |         |      |
    |        |                                  |        |<+ (5) +>|      |
    |        |                                  |        |Completed|      |
    |        |                                  |        |         |      |
    |        |--(8)--- Continue Request (B) --->|        |         +------+
    |        |                                  |        |
    |        |<-(9)------ Grant Access ---------|        |
    |        |                                  |        |
    +--------+                                  +--------+
```

1. The RC requests access to the resource (Section 2). The RC does not send any interactions modes 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 RO, the AS indicates to the RO 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). Note that the AS may need to determine that the RO has not approved the request in a sufficient amount of time and return an appropriate error to the RC.

8. The RC continues to poll the AS (Section 5.2) with the new continuation information from (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.

1.4.4. 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. Since there is no explicit RO, the RC does not interact with an RO.

```
       +--------+                                  +--------+
       |   RC   |                                  |   AS   |
       |        |--(1)--- Request Access --------->|        |
       |        |                                  |        |
       |        |<-(2)---- Grant Access -----------|        |
       |        |                                  |        |
       +--------+                                  +--------+
```

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

1.4.5. 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.

```
       +--------+                                          +--------+
       |   RC   |                                          |   AS   |
       |        |--(1)--- Request Access ----------------->|        |
       |        |                                          |        |
       |        |<-(2)--- Grant Access --------------------|        |
       |        |                                          |        |
       |        |                             +--------+   |        |
       |        |--(3)--- Access Resource --->|   RS   |   |        |
       |        |                             |        |   |        |
       |        |<-(4)--- Error Response -----|        |   |        |
       |        |                             +--------+   |        |
       |        |                                          |        |
       |        |--(5)--- Rotate Token ------------------->|        |
       |        |                                          |        |
       |        |<-(6)--- Rotated Token -------------------|        |
       |        |                                          |        |
       +--------+                                          +--------+
```

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 RC sends JSON [RFC8259] document with an object as its root. Each member of the request object represents a different aspect of the RC's request. Each field is described in detail in a section below.

resources (object / array of objects/strings) Describes the rights that the RC is requesting for one or more access tokens to be used at RS's. Section 2.1

subject (object) Describes the information about the RO that the RC is requesting to be returned directly in the response from the AS. Section 2.2

client (object / string) Describes the RC that is making this request, including the key that the RC will use to protect this request and any continuation requests at the AS and any user-facing information about the RC used in interactions at the AS. Section 2.3

user (object / string) Identifies the RQ to the AS in a manner that the AS can verify, either directly or by interacting with the RQ to determine their status as the RO. Section 2.4

interact (object) Describes the modes that the RC has for allowing the RO to interact with the AS and modes for the RC to receive updates when interaction is complete. Section 2.5

capabilities (array of strings) Identifies named extension capabilities that the RC can use, signaling to the AS which extensions it can use. Section 2.6

existing_grant (string) Identifies a previously-existing grant that the RC is extending with this request. Section 2.7

claims (object) Identifies the identity claims to be returned as part of an OpenID Connect claims request. Section 2.8

Additional members of this request object can be defined by extensions to this protocol as described in Section 2.9

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"
    ],
    "client": {
      "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
      },
      "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"
        }
    },
    "capabilities": ["ext1", "ext2"],
    "subject": {
        "sub_ids": ["iss_sub", "email"],
        "assertions": ["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 RC is requesting one or more access tokens for the purpose of accessing an API, the RC MUST include a "resources" field. This field MUST be an array (for a single access token (Section 2.1.1)) or an object (for multiple access tokens (Section 2.1.3)), as described in the following sections.

2.1.1. Requesting a Single Access Token

When requesting an access token, the RC MUST send a "resources" field containing a JSON array. The elements of the JSON array represent rights of access that the RC is requesting in the access token. The requested access is the sum of all elements within the array.

The RC 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 RC is calling.

type (string) The type of resource request as a string. This field
   MAY define which other fields are allowed in the request object.
   This field is REQUIRED.

The value of this field is under the control of the AS. This field MUST be compared using an exact byte match of the string value against known types by the AS. The AS MUST ensure that there is no collision between different authorization data types that it supports. The AS MUST NOT do any collation or normalization of data types during comparison. It is RECOMMENDED that designers of general-purpose APIs use a URI for this field to avoid collisions between multiple API types protected by a single AS.

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. The available values for these properties are determined by the API being protected at the RS.

[[See issue #34 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/34)]]

actions (array of strings) The types of actions the RC will take at the RS as an array of strings. For example, an RC asking for a combination of "read" and "write" access.

locations (array of strings) The location of the RS as an array of strings. These strings are typically URIs identifying the location of the RS.

datatypes (array of strings) The kinds of data available to the RC at the RS's API as an array of strings. For example, an RC asking for access to raw "image" data and "metadata" at a photograph API.

identifier (string) A string identifier indicating a specific resource at the RS. For example, a patient identifier for a medical API or a bank account number for a financial API.

The following non-normative example shows the use of both common and API-specific fields as part of two different access "type" values.

```
    "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"
        }
    ]
```

If this request is approved, the resulting access token (Section 3.2.1) will include the sum of both of the requested types of access.

2.1.2. Requesting Resources By Reference

Instead of sending an object describing the requested resource (Section 2.1.1), a RC 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.

[[See issue #35 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/35)]]

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

This value is opaque to the RC and MAY be any valid JSON string, and therefore could include spaces, unicode characters, and properly escaped string sequences. However, in some situations the value is intended to be seen and understood be the RC developer. In such cases, the API designer choosing any such human-readable strings SHOULD take steps to ensure the string values are not easily confused by a developer

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 RC. 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"
    ]
```

[[See issue #36 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/36)]]

2.1.3. Requesting Multiple Access Tokens

When requesting multiple access tokens, the resources field is a JSON object. The names of the JSON object fields are token identifiers chosen by the RC, and MAY be any valid string. The values of the JSON object fields 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"
                   ]
               }
           ]
       }
```

Any approved access requests are returned in the multiple access token response (Section 3.2.2) structure using the token identifiers in the request.

2.1.4. Signaling Token Behavior

While the AS is ultimately in control of how tokens are returned and bound to the RC, sometimes the RC has context about what it can support that can affect the AS's response. This specification defines several flags that are passed as resource reference strings (Section 2.1.2).

Each flag applies only to the single resource request in which it appears.

Support of all flags is optional, such as any other resource reference value.

multi_token The RC wishes to support multiple simultaneous access tokens through the token rotation process. When the RC rotates an access token (Section 6.1), the AS does not invalidate the previous access token. The old access token continues to remain valid until such time as it expires or is revoked through other means.

split_token The RC is capable of receiving multiple access tokens (Section 3.2.2) in response to any single token request (Section 2.1.1), or receiving a different number of tokens than specified in the multiple token request (Section 2.1.3). The labels of the returned additional tokens are chosen by the AS. The RC MUST be able to tell from the token response where and how it can use each of the access tokens. [[See issue #37 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/37)]]

bind_token The RC wants the issued access token to be bound to the key the RC used (Section 2.3.2) to make the request. The resulting access token MUST be bound using the same "proof" mechanism used by the client with a "key" value of "true", indicating the client's presented key is to be used for binding. [[See issue #38 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/38)]]

The AS MUST respond with any applied flags in the token response (Section 3.2) "resources" section.

In this non-normative example, the requested access token is to be bound to the client's key and should be kept during rotation.

```
       "resources": [
           {
               "type": "photo-api",
               "actions": [
                   "read",
                   "write",
                   "dolphin"
               ],
               "locations": [
                   "https://server.example.net/",
                   "https://resource.local/other"
               ],
               "datatypes": [
                   "metadata",
                   "images"
               ]
           },
           "read",
           "bind_token",
           "multi_token"
       ]
```

Additional flags can be registered in a registry TBD (Section 12).

[[See issue #39 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/39)]]

2.2. Requesting User Information

If the RC is requesting information about the RO from the AS, it sends a "subject" field as a JSON object. This object MAY contain the following fields (or additional fields defined in a registry TBD (Section 12)).

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

assertions (array of strings) An array of requested assertion formats. Possible values include "id_token" for an [OIDC] ID Token and "saml2" for a SAML 2 assertion. Additional assertion values are defined by a registry TBD (Section 12). [[See issue #41 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/41)]]

```
   "subject": {
      "sub_ids": [ "iss_sub", "email" ],
      "assertions": [ "id_token", "saml2" ]
   }
```

The AS can determine the RO's identity and permission for releasing this information through interaction with the RO (Section 4), AS policies, or assertions presented by the RC (Section 2.4). If this is determined positively, the AS MAY return the RO's information in its response (Section 3.4) as requested.

Subject identifiers requested by the RC serve only to identify the RO in the context of the AS and can't be used as communication channels by the RC, as discussed in Section 3.4. One method of requesting communication channels and other identity claims are discussed in Section 2.8.

The AS SHOULD NOT re-use subject identifiers for multiple different ROs.

[[See issue #42 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/42)]]

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

[[See issue #43 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/43)]]

2.3. Identifying the RC

When sending a non-continuation request to the AS, the RC MUST identify itself by including the "client" field of the request and by signing the request as described in Section 8. Note that for a continuation request (Section 5), the RC instance is identified by its association with the request being continued and so this field is not sent under those circumstances.

When RC information is sent by value, the "client" field of the request consists of a JSON object with the following fields.

key (object / string) The public key of the RC to be used in this request as described in Section 2.3.2. This field is REQUIRED.

class_id (string) An identifier string that the AS can use to identify the software comprising this instance of the RC. The contents and format of this field are up to the AS. This field is OPTIONAL.

display (object) An object containing additional information that the AS MAY display to the RO during interaction, authorization, and management. This field is OPTIONAL.

```
"client": {
    "key": {
        "proof": "httpsig",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
                    "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_JtffXyaSx8xY..."
        },
        "cert": "MIIEHDCCAwSgAwIBAgIBATANBgkqhkiG9w0BAQsFA..."
    },
    "class_id": "web-server-1234",
    "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
    }
}
```

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

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 of methods is described in Section 8.

Note that the AS MAY know the RC’s public key ahead of time, and the AS MAY apply different policies to the request depending on what has been registered against that key. If the same public key is sent by value on subsequent access requests, the AS SHOULD treat these requests as coming from the same RC software instance for purposes of identification, authentication, and policy application. If the AS does not know the RC’s public key ahead of time, the AS MAY accept or reject the request based on AS policy, attestations within the client request, and other mechanisms.

[[See issue #44 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/44)]]

2.3.1. Identifying the RC Instance

If the RC has an instance identifier that the AS can use to determine appropriate key information, the RC can send this value in the "instance_id" field. The instance identifier MAY be assigned to an RC instance at runtime through the Section 3.5 or MAY be obtained in another fashion, such as a static registration process at the AS.

instance_id (string) An identifier string that the AS can use to identify the particular instance of this RC. The content and structure of this identifier is opaque to the RC.

```
"client": {
    "instance_id": "client-541-ab"
}
```

If there are no additional fields to send, the RC MAY send the instance identifier as a direct reference value in lieu of the object.

```
"client": "client-541-ab"
```

When the AS receives a request with an instance identifier, the AS MUST ensure that the key used to sign the request (Section 8) is associated with the instance identifier.

If the "instance_id" field is sent, it MUST NOT be accompanied by other fields unless such fields are explicitly marked safe for inclusion alongside the instance identifier.

[[See issue #45 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/45)]]

If the AS does not recognize the instance identifier, the request MUST be rejected with an error.

If the RC instance is identified in this manner, the registered key for the RC MAY be a symmetric key known to the AS. The RC MUST NOT send a symmetric key by value in the request, as doing so would expose the key directly instead of proving possession of it.

[[See issue #46 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/46)]]

2.3.2. Identifying the RC Key

The RC key MUST be a public key in at least one supported format and MUST be applicable to 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 (string) 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 (object) 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 (string) PEM serialized value of the certificate used to sign the request, with optional internal whitespace.

cert#S256 (string) The certificate thumbprint calculated as per OAuth-MTLS [RFC8705] in base64 URL encoding.

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

[[See issue #47 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/47)]]

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

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

Continuation requests (Section 5) MUST use the same key (or its most recent rotation) and proof method as the initial request.

2.3.3. Providing Displayable RC Information

If the RC has additional information to display to the RO 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 RO during any interactive sequences.

name (string) Display name of the RC software

uri (string) User-facing web page of the RC software

logo_uri (string) Display image to represent the RC software

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

[[See issue #48 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/48)]]

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

The AS SHOULD use these values during interaction with the RO. The values are for informational purposes only and MUST NOT be taken as authentic proof of the RC's identity or source. The AS MAY restrict display values to specific RC instances, as identified by their keys in Section 2.3.

2.3.4. Authenticating the RC

If the presented key is known to the AS and is associated with a single instance of the RC software, the process of presenting a key and proving possession of that key is sufficient to authenticate the RC to the AS. The AS MAY associate policies with the RC software identified by this key, such as limiting which resources can be requested and which interaction methods can be used. For example, only specific RCs with certain known keys might be trusted with access tokens without the AS interacting directly with the RO as in Appendix D.

The presentation of a key allows the AS to strongly associate multiple successive requests from the same RC with each other. This is true when the AS knows the key ahead of time and can use the key to authenticate the RC software, but also if the key is ephemeral and created just for this series of requests. As such the AS MAY allow for RCs to make requests with unknown keys. This pattern allows for ephemeral RCs, such as single-page applications, and RCs with many individual instances, 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 RCs with unknown keys. For example, the AS could have a policy saying that only previously-registered RCs can request particular resources, or that all RCs with unknown keys have to be interactively approved by an RO.

2.4. Identifying the User

If the RC knows the identity of the RQ through one or more identifiers or assertions, the RC MAY send that information to the AS in the "user" field. The RC MAY pass this information by value or by reference.

sub_ids (array of strings) An array of subject identifiers for the RQ, as defined by [I-D.ietf-secevent-subject-identifiers].

assertions (object) An object containing assertions as values keyed on the assertion type defined by a registry TBD (Section 12). Possible keys include "id_token" for an [OIDC] ID Token and "saml2" for a SAML 2 assertion. Additional assertion values are defined by a registry TBD (Section 12). [[See issue #41 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/41)]]

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

Subject identifiers are hints to the AS in determining the RO and MUST NOT be taken as declarative statements that a particular RO is present at the RC and acting as the RQ. Assertions SHOULD be validated by the AS. [[See issue #49 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/49)]]

If the identified RQ does not match the RO present at the AS during an interaction step, the AS SHOULD reject the request with an error.

[[See issue #50 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/50)]]

If the AS trusts the RC to present verifiable assertions, the AS MAY decide, based on its policy, to skip interaction with the RO, even if the RC provides one or more interaction modes in its request.

2.4.1. Identifying the User by Reference

User reference identifiers can be dynamically issued by the AS (Section 3.5) to allow the RC to represent the same RQ to the AS over subsequent requests.

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

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

User reference identifiers are not intended to be human-readable user identifiers or structured assertions. For the RC to send either of these, use the full user request object (Section 2.4) instead.

[[See issue #51 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/51)]]

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

2.5. Interacting with the User

Many times, the AS will require interaction with the RO in order to approve a requested delegation to the RC for both resources and direct claim information. Many times the RQ using the RC is the same person as the RO, and the RC can directly drive interaction with the AS by redirecting the RQ on the same device, or by launching an application. Other times, the RC can provide information to start the RO’s interaction on a secondary device, or the RC will wait for the RO to approve the request asynchronously. The RC could also be signaled that interaction has completed by the AS making callbacks. To facilitate all of these modes, the RC declares the means that it can interact using the "interact" field.

The "interact" field is a JSON object with keys that declare different interaction modes. A RC MUST NOT declare an interaction mode it does not support. The RC MAY send multiple modes 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 modes (Section 3.3) in a request, depending on its capabilities and what is allowed to fulfill the request. This specification defines the following interaction modes:

redirect (boolean) Indicates that the RC can direct the RQ to an arbitrary URL at the AS for interaction. Section 2.5.1

app (boolean) Indicates that the RC can launch an application on the RQ’s device for interaction. Section 2.5.2

callback (object) Indicates that the RC can receive a callback from the AS after interaction with the RO has concluded. Section 2.5.3

user_code (boolean) Indicates that the RC can communicate a human-readable short code to the RQ for use with a stable URL at the AS. Section 2.5.4

ui_locales (array of strings) Indicates the RQ’s preferred locales that the AS can use during interaction, particularly before the RO has authenticated. Section 2.5.5

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

[[See issue #52 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/52)]]

In this non-normative example, the RC is indicating that it can redirect (Section 2.5.1) the RQ to an arbitrary URL and can receive a callback (Section 2.5.3) through a browser request.

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

In this non-normative example, the RC is indicating that it can display a user code (Section 2.5.4) and direct the RQ to an arbitrary URL of maximum length (Section 2.5.1.1) 255 characters, but it cannot accept a callback.

```
"interact": {
    "redirect": 255,
    "user_code": true
}
```

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.

The AS SHOULD apply suitable timeouts to any interaction mechanisms provided, including user codes and redirection URLs. The RC SHOULD apply suitable timeouts to any callback URLs.

2.5.1. Redirect to an Arbitrary URL

If the RC is capable of directing the RQ to a URL defined by the AS at runtime, the RC indicates this by sending the "redirect" field with the boolean value "true". The means by which the RC will activate this URL is out of scope of this specification, but common methods include an HTTP redirect, launching a browser on the RQ's device, providing a scannable image encoding, and printing out a URL to an interactive console.

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

If this interaction mode is supported for this RC 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 RC would prefer to redirect to a shortened URL defined by the AS at runtime, the RC 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. [[See issue #53 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/53)]]

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

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

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

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

[[See issue #54 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/54)]]

2.5.3. Receive a Callback After Interaction

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

uri (string) 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 RO'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 use a unique callback URI to link to that ongoing state. 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. [[See issue #55 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/55)]]

nonce (string) 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 (string) REQUIRED. The callback method that the AS will use to contact the RC. 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 (string) 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". [[See issue #56 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/56)]]

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

If this interaction mode is supported for this RC 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.

[[See issue #58 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/58)]]

[[See issue #59 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/59)]]

2.5.3.1. Receive an HTTP Callback Through the Browser

A callback "method" value of "redirect" indicates that the RC will expect a call from the RO'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 by the RC as described in Section 4.4.1.

Since the incoming request to the callback URL is from the RO's browser, this method is usually used when the RO and RQ are the same entity. As such, the RC MUST ensure the RQ is present on the request to prevent substitution attacks.

2.5.3.2. Receive an HTTP Direct Callback

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

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

Requests to the callback URI MUST be processed by the RC as described in Section 4.4.2.

Since the incoming request to the callback URL is from the AS and not from the RO's browser, the RC MUST NOT require the RQ to be present on the incoming HTTP request.

[[See issue #60 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/60)]]

2.5.4. Display a Short User Code

If the RC is capable of displaying or otherwise communicating a short, human-entered code to the RO, the RC 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, as described in Section 3.3.4.

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

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

2.5.5. Indicate Desired Interaction Locales

If the RC knows the RQ's locale and language preferences, the RC can send this information to the AS using the "ui_locales" field with an array of locale strings as defined by [RFC5646].

```
"interact": {
    "ui_locales": ["en-US", "fr-CA"]
}
```

If possible, the AS SHOULD use one of the locales in the array, with preference to the first item in the array supported by the AS. If none of the given locales are supported, the AS MAY use a default locale.

2.5.6. Extending Interaction Modes

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

[[See issue #61 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/61)]]

2.6. Declaring RC Capabilities

If the RC 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"]
```

2.7. Referencing an Existing Grant Request

If the RC has a reference handle from a previously granted request, it MAY send that reference in the "existing_grant" field. This field is a single string consisting of the "value" of the "access_token" returned in a previous request’s continuation response (Section 3.1).

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

The AS MUST dereference the grant associated with the reference and process this request in the context of the referenced one. The AS MUST NOT alter the existing grant associated with the reference.

[[See issue #62 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/62)]]

2.8. Requesting OpenID Connect Claims

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

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

The contents of the "claims" parameter have the same semantics as they do in OpenID Connect's "claims" authorization request parameter, 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 "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 a returned 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.

[[See issue #63 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/63)]]

[[See issue #64 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/64)]]

2.9. 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 extension item affects or influences the values or behavior of other request and response objects.

[[See issue #65 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/65)]]

3. Grant Response

In response to a RC's request, the AS responds with a JSON object as the HTTP entity body. Each possible field is detailed in the sections below continue (object) Indicates that the RC can continue the request by making an additional request using these parameters. Section 3.1

access_token (object) A single access token that the RC can use to call the RS on behalf of the RO. Section 3.2.1

multiple_access_token (object) Multiple named access tokens that the RC can use to call the RS on behalf of the RO. Section 3.2.2

interact (object) Indicates that interaction through some set of defined mechanisms needs to take place. Section 3.3

subject (object) Claims about the RO as known and declared by the AS. Section 3.4

instance_id (string) An identifier this RC instance can use to identify itself when making future requests. Section 3.5

user_handle (string) An identifier this RC instance can use to identify its current RQ when making future requests. Section 3.5

error (object) An error code indicating that something has gone wrong. Section 3.6

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/4CF492MLVMSW9MKMXKHQ",
        "callback": "MBDOFXG4Y5CVJCX821LH"
    },
    "continue": {
        "access_token": {
            "value": "80UPRY5NM33OMUKMKSKU",
            "key": true
        },
        "uri": "https://server.example.com/tx"
    }
}
```

In this example, the AS is returning a bearer access token (Section 3.2.1) with a management URL and a subject identifier (Section 3.4) in the form of an email address.

```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L"
    },
    "subject": {
        "sub_ids": [ {
           "subject_type": "email",
           "email": "user@example.com",
        } ]
    }
}
```

3.1. Request Continuation

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.

uri (string) REQUIRED. The URI at which the RC can make continuation requests. This URI MAY vary per request, or MAY be stable at the AS if the AS includes an access token. The RC MUST use this value exactly as given when making a continuation request (Section 5).

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

access_token (object) RECOMMENDED. A unique access token for continuing the request, in the format specified in Section 3.2.1. This access token MUST be bound to the RC’s key used in the request and MUST NOT be a "bearer" token. This access token MUST NOT be usable at resources outside of the AS. If the AS includes an access token, the RC MUST present the access token in all requests to the continuation URI as described in Section 7. [[See issue #66 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/66)]]

```
{
    "continue": {
        "access_token": {
            "value": "80UPRY5NM33OMUKMKSKU",
            "key": true
        },
        "uri": "https://server.example.com/continue",
        "wait": 60
    }
}
```

The RC can use the values of this field to continue the request as described in Section 5. Note that the RC MUST sign all continuation requests with its key as described in Section 8. If the AS includes an "access_token", the RC MUST present the access token in its continuation request.

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

[[See issue #67 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/67)]]

3.2. Access Tokens

If the AS has successfully granted one or more access tokens to the RC, the AS responds with either the "access_token" or the "multiple_access_token" field. The AS MUST NOT respond with both the "access_token" and "multiple_access_token" fields.

[[See issue #68 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/68)]]

3.2.1. Single Access Token

If the RC 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 (string) REQUIRED. The value of the access token as a string. The value is opaque to the RC. The value SHOULD be limited to ASCII characters to facilitate transmission over HTTP headers within other protocols without requiring additional encoding.

manage (string) OPTIONAL. The management URI for this access token.

If provided, the RC MAY manage its access token as described in Section 6. This management URI is a function of the AS and is separate from the RS the RC is requesting access to. This URI MUST NOT include the access token value and SHOULD be different for each access token issued in a request.

resources (array of objects/strings) RECOMMENDED. A description of the rights associated with this access token, as defined in Section 2.1.1. If included, this MUST reflect the rights associated with the issued access token. These rights MAY vary from what was requested by the RC.

expires_in (integer) OPTIONAL. The number of seconds in which the access will expire. The RC MUST NOT use the access token past this time. An RS MUST NOT accept an access token past this time. Note that the access token MAY be revoked by the AS or RS at any point prior to its expiration.

key (object / string / boolean) REQUIRED. The key that the token is bound to. If the boolean value "true" is used, the token is bound to the key used by the RC (Section 2.3.2) in its request for access. If the boolean value "false" is used, the token is a bearer token with no key bound to it. Otherwise, the key MUST be an object or string in a format described in Section 2.3.2, describing a public key to which the RC can use the associated private key. The RC MUST be able to dereference or process the key information in order to be able to sign the request.

The following non-normative example shows a single bearer token with a management URL that has access to three described resources.

```
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L",
        "resources": [
            {
                "type": "photo-api",
                "actions": [
                    "read",
                    "write",
                    "dolphin"
                ],
                "locations": [
                    "https://server.example.net/",
                    "https://resource.local/other"
                ],
                "datatypes": [
                    "metadata",
                    "images"
                ]
            },
            "read", "dolphin-metadata"
        ]
    }
```

The following non-normative example shows a single access token bound to the RC’s key, which was presented using the detached JWS (Section 8.1) binding method.

```
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "key": true,
        "resources": [
            "finance", "medical"
        ]
    }
```

If the RC requested multiple access tokens (Section 2.1.3), the AS MUST NOT respond with a single access token structure unless the RC sends the "split_token" flag as described in Section 2.1.4.

[[See issue #69 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/69)]]

3.2.2. Multiple Access Tokens

If the RC 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 RC 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.

In this non-normative example, two bearer tokens are issued under the names "token1" and "token2", and only the first token has a management URL associated with it.

```
    "multiple_access_tokens": {
        "token1": {
            "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
            "key": false,
            "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CD
FONP219RP1L"
        },
        "token2": {
            "value": "UFGLO2FDAFG7VGZZPJ3IZEMN21EVU71FHCARP4J1",
            "key": false
        }
    }
```

Each access token corresponds to the named resources arrays in the RC's request (Section 2.1.3).

The multiple access token response MUST be used when multiple access tokens are requested, even if only one access token is issued as a result of the request. The AS MAY refuse to issue one or more of the requested access tokens, for any reason. In such cases the refused token is omitted from the response and all of the other issued access tokens are included in the response the requested names appropriate names.

If the RC requested a single access token (Section 2.1.1), the AS MUST NOT respond with the multiple access token structure unless the RC sends the "split_token" flag as described in Section 2.1.4.

Each access token MAY have different proofing mechanisms. If management is allowed, each access token SHOULD have different management URIs.

[[See issue #70 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/70)]]

3.3. Interaction Modes

If the RC has indicated a capability to interact with the RO in its request (Section 2.5), and the AS has determined that interaction is both supported and necessary, the AS responds to the RC with any of the following values in the "interact" field of the response. There is no preference order for interaction modes in the response, and it is up to the RC to determine which ones to use. All supported interaction methods are included in the same "interact" object.

redirect (string) Redirect to an arbitrary URL. Section 3.3.1

app (string) Launch of an application URL. Section 3.3.2

callback (string) Callback to an RC URL after interaction is completed. Section 3.3.3

user_code (object) Display a short user code. Section 3.3.4

Additional interaction mode responses can be defined in a registry TBD (Section 12).

The AS MUST NOT respond with any interaction mode that the RC did not indicate in its request. The AS MUST NOT respond with any interaction mode that the AS does not support. Since interaction responses include secret or unique information, the AS SHOULD respond to each interaction mode only once in an ongoing request, particularly if the RC modifies its request (Section 5.3).

3.3.1. Redirection to an arbitrary URL

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

```
"interact": {
    "redirect": "https://interact.example.com/4CF492MLVMSW9MKMXKHQ"
}
```

The interaction URL returned represents a function of the AS but MAY be completely distinct from the URL the RC uses to request access (Section 2), allowing an AS to separate its user-interactive functionality from its back-end security functionality.

[[See issue #72 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/72)]]

The RC sends the RQ to the URL to interact with the AS. The RC MUST NOT alter the URL in any way. The means for the RC to send the RQ 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 RC indicates that it can launch an application URL (Section 2.5.2) and the AS supports this mode for the RC's request, the AS responds with the "app" field, which is a string containing the URL to direct the RQ 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 RC launches the URL as appropriate on its platform, and the means for the RC to launch this URL is out of scope of this specification. The RC MUST NOT alter the URL in any way. The RC MAY attempt to detect if an installed application will service the URL being sent before attempting to launch the application URL.

[[See issue #71 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/71)]]

3.3.3. Post-interaction Callback to an RC URL

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

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

[[See issue #73 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/73)]]

When the RO completes interaction at the AS, the AS MUST call the RC'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 RC 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 RC indicates that it can display a short user-typeable code (Section 2.5.4) and the AS supports this mode for the RC's request, the AS responds with a "user_code" field. This field is an object that contains the following members.

code (string) 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 (string) RECOMMENDED. The interaction URL that the RC will direct the RO to. This URL MUST be stable at the AS such that RCs can be statically configured with it.

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

The RC MUST communicate the "code" to the RQ in some fashion, such as displaying it on a screen or reading it out audibly. The "code" is a one-time-use credential that the AS uses to identify the pending request from the RC. When the RO enters this code (Section 4.2) 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. If the AS detects too many unrecognized codes entered, it SHOULD display an error to the user.

The RC SHOULD also communicate the URL if possible to facilitate user interaction, but since the URL should be stable, the RC should be able to safely decide to not display this value. As this interaction mode is designed to facilitate interaction via a secondary device, it is not expected that the RC redirect the RQ to the URL given here at runtime. Consequently, the URL needs to be stable enough that a RC could be statically configured with it, perhaps referring the RQ to the URL via documentation instead of through an interactive means. If the RC is capable of communicating an arbitrary URL to the RQ, such as through a scannable code, the RC can use the "redirect" (Section 2.5.1) mode for this purpose instead of or in addition to the user code mode.

The interaction URL returned represents a function of the AS but MAY be completely distinct from the URL the RC uses to request access (Section 2), allowing an AS to separate its user-interactive functionality from its back-end security functionality.

[[See issue #72 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/72)]]

3.3.5. Extending Interaction Mode Responses

Extensions to this specification can define new interaction mode responses in a registry TBD (Section 12). Extensions MUST document the corresponding interaction request.

3.4. Returning User Information

If information about the RO is requested and the AS grants the RC 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 (array of strings) An array of subject identifiers for the RO, as defined by [I-D.ietf-secevent-subject-identifiers]. [[See issue #74 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/74)]]

assertions (object) An object containing assertions as values keyed on the assertion type defined by a registry TBD (Section 12). [[See issue #41 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/41)]]

updated_at (string) Timestamp as an ISO8610 date string, indicating when the identified account was last updated. The RC MAY use this value to determine if it needs to request updated profile information through an identity API. The definition of such an identity API is out of scope for this specification.

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

The AS MUST return the "subject" field only in cases where the AS is sure that the RO and the RQ are the same party. This can be accomplished through some forms of interaction with the RO (Section 4).

Subject identifiers returned by the AS SHOULD uniquely identify the RO at the AS. Some forms of subject identifier are opaque to the RC (such as the subject of an issuer and subject pair), while others forms (such as email address and phone number) are intended to allow the RC to correlate the identifier with other account information at the RC. The RC MUST NOT request or use any returned subject identifiers for communication purposes (see Section 2.2). That is, a subject identifier returned in the format of an email address or a phone number only identifies the RO to the AS and does not indicate that the AS has validated that the represented email address or phone number in the identifier is suitable for communication with the current user. To get such information, the RC MUST use an identity protocol to request and receive additional identity claims. While Section 2.8 specifies one such method, other identity protocols could also be used on top of GNAP to convey this information and the details of an identity protocol and associated schema are outside the scope of this specification.

[[See issue #75 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/75)]]

[[See issue #74 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/74)]]

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 RC's request can be passed as either a value or a reference. The use of a reference in place of a value allows for a client to optimize requests to the AS.

Some references, such as for the RC instance's identity (Section 2.3.1) or the requested resources (Section 2.1.2), can be managed statically through an admin console or developer portal provided by the AS or RS. The developer of the RC can include these values in their code for a more efficient and compact request.

If desired, the AS MAY also generate and return some of these references dynamically to the RC in its response to facilitate multiple interactions with the same software. The RC 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 RC as secrets. Handle values MUST be unguessable and MUST NOT contain any sensitive information. Handle values are opaque to the RC.

[[See issue #76 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/76)]]

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).

instance_id (string) A string value used to represent the information in the "client" object that the RC can use in a future request, as described in Section 2.3.1.

user_handle (string) A string value used to represent the current user. The RC 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",
    "instance_id": "7C7C4AZ9KHRS6X63AJAO",
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "key": false
    }
}
```

[[See issue #77 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/77)]]

[[See issue #78 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/78)]]

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 (string) 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_request The request referenced an unknown ongoing access request.

[[See issue #79 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/79)]]

3.7. Extending the Response

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

[[See issue #80 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/80)]]

4. Interaction at the AS

If the RC indicates that it is capable of driving interaction with the RO in its request (Section 2.5), and the AS determines that interaction is required and responds to one or more of the RC's interaction modes, the RC SHOULD initiate one of the returned interaction modes 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 current user (who may be the RQ) as the RO

* gather consent and authorization from the RO for access to requested resources and direct information

* allow the RO to modify the parameters of the request (such as disallowing some requested resources or specifying an account or record)

* provide warnings to the RO about potential attacks or negative effects of the requested information

[[See issue #81 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/81)]]

4.1. Interaction at a Redirected URI

When the RO is directed to the AS through the "redirect" (Section 3.3.1) mode, the AS can interact with the RO through their web browser to authenticate the user as an RO and gather their consent. Note that since the RC 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 RO and MUST NOT attempt to redirect the RO back to any RC even if a callback is supplied (Section 2.5.3).

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.

With this method, it is common for the RO to be the same party as the RQ, since the RC has to communicate the redirection URI to the RQ.

4.2. Interaction at the User Code URI

When the RO is directed to the AS through the "user_code" (Section 3.3.4) mode, the AS can interact with the RO 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 RO and MUST NOT attempt to redirect the RO back to any RC even if a callback is supplied (Section 2.5.3).

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.

While it is common for the RO to be the same party as the RQ, since the RC has to communicate the user code to someone, there are cases where the RQ and RO are separate parties and the authorization happens asynchronously.

4.3. Interaction through an Application URI

When the RC successfully launches an application through the "app" mode (Section 3.3.2), the AS interacts with the RO 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.

[[See issue #82 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/82)]]

4.4. Post-Interaction Completion

Upon completing an interaction with the RO, if a "callback" (Section 3.3.3) mode is available with the current request, the AS MUST follow the appropriate method at the end of interaction to allow the RC to continue. If this mode is not available, the AS SHOULD instruct the RO to return to their RC software upon completion. Note that these steps still take place in most error cases, such as when the RO has denied access. This pattern allows the RC to potentially recover from the error state without restarting the request from scratch by modifying its request or providing additional information directly to the AS.

[[See issue #83 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/83)]]

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 interaction reference MUST be one-time-use.

The AS MUST calculate a hash value based on the RC and AS nonces and the interaction reference, as described in Section 4.4.3. The RC will use this value to validate the return call from the AS.

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

4.4.1. Completing Interaction with a Browser Redirect to the Callback URI

When using the "callback" interaction mode (Section 3.3.3) with the "redirect" method, the AS signals to the RC that interaction is complete and the request can be continued by directing the RO (in their browser) back to the RC'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 RC'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 RO to this URL are outside the scope of this specification, but common options include redirecting the RO from a web page and launching the system browser with the target URL.

```
https://client.example.net/return/123455
  ?hash=p28jsq0Y2KK3WS__a42tavNC64ldGTBroywsWxT4md_jZQ1R2HZT8BOWYHcLmObM7XHPAdJzT
ZMtKBsaraJ64A
  &interact_ref=4IFWWIKYBC2PQ6U56NL1
```

When receiving the request, the RC MUST parse the query parameters to calculate and validate the hash value as described in Section 4.4.3. If the hash validates, the RC 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 Direct HTTP Request Callback

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

hash (string) REQUIRED. The interaction hash value as described in Section 4.4.3.

interact_ref (string) 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_jZQ1R2HZT8BOWYHcLmObM7XHPAd
JzTZMtKBsaraJ64A",
  "interact_ref": "4IFWWIKYBC2PQ6U56NL1"
}
```

When receiving the request, the RC MUST parse the JSON object and validate the hash value as described in Section 4.4.3. If the hash validates, the RC 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 RC's callback URL ties the front channel response to an ongoing request by using values known only to the parties involved. This security mechanism allows the RC to protect itself against several kinds of session fixation and injection attacks. The AS MUST always provide this hash, and the RC MUST validate the hash when received.

[[See issue #84 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/84)]]

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 RC'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".

[[See issue #56 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/56)]]

4.4.3.1. SHA3-512

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_jZQ1R2HZT8BOWYHcLmObM7XHPAdJzTZMtKBsar
aJ64A
```

4.4.3.2. SHA2-512

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_Bpj84rh4mC9aE9x7HPB
FcIHw
```

5. Continuing a Grant Request

While it is possible for the AS to return a Section 3 with all the RC's requested information (including access tokens (Section 3.2) and direct user information (Section 3.4)), it's more common that the AS and the RC will need to communicate several times over the lifetime of an access grant. This is often part of facilitating interaction (Section 4), but it could also be used to allow the AS and RC to continue negotiating the parameters of the original grant request (Section 2).

To enable this ongoing negotiation, the AS returns a "continue" field in the response (Section 3.1) that contains information the RC needs to continue this process with another request, including a URI to access as well as an optional access token to use during the continued requests.

When the RC makes any calls to the continuation URL, the RC MUST present proof of the most recent key associated with this ongoing request by signing the request as described in Section 8. The key in use will be either the key from the initial request (Section 2.3.2) or its most recent rotation. [[See issue #85 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/85)]]

For example, here the RC makes a POST request and signs with detached JWS:

```
POST /continue/80UPRY5NM33OMUKMKSKU HTTP/1.1
Host: server.example.com
Detached-JWS: ejy0...
```

If the AS includes an "access_token" in the "continue" response in Section 3.1, the RC MUST include the access token the request as described in Section 7. Note that the access token is always bound to the RC's presented key (or its most recent rotation).

For example, here the RC makes a POST request with the interaction reference, includes the access token, and signs with detached JWS:

```
POST /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...

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

The AS MUST be able to tell from the RC's request which specific ongoing request is being accessed. Common methods for doing so include using a unique, unguessable URL for each continuation response, associating the request with the provided access token, or allowing only a single ongoing grant request for a given RC instance at a time. If the AS cannot determine a single active grant request to map the continuation request to, the AS MUST return an error.

The ability to continue an already-started request allows the RC to perform several important functions, including presenting additional information from interaction, modifying the initial request, and getting the current state of the request.

If a "wait" parameter was included in the continuation response (Section 3.1), the RC MUST NOT call the continuation URI prior to waiting the number of seconds indicated. If no "wait" period is indicated, the RC SHOULD wait at least 5 seconds If the RC does not respect the given wait period, the AS MUST return an error. [[See issue #86 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/86)]]

The response from the AS is a JSON object and MAY contain any of the fields described in Section 3, as described in more detail in the sections below.

If the AS determines that the RC can make a further continuation request, the AS MUST include a new "continue" response (Section 3.1). If the continuation was previously bound to an access token, the new "continue" response MUST include a bound access token as well, and this token SHOULD be a new access token. If the AS does not return a new "continue" response, the RC MUST NOT make an additional continuation request. If a RC does so, the AS MUST return an error. [[See issue #87 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/87)]]

For continuation functions that require the RC to send a message body, the body MUST be a JSON object.

5.1. Continuing After a Completed Interaction

When the AS responds to the RC's "callback" parameter as in Section 4.4.1, this response includes an interaction reference. The RC MUST include that value as the field "interact_ref" in a POST request to the continuation URI.

```
POST /continue/80UPRY5NM33OMUKMKSKU HTTP/1.1
Host: server.example.com
Content-type: application/json
Detached-JWS: ejy0...

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

Since the interaction reference is a one-time-use value as described in Section 4.4.1, if the RC needs to make additional continuation calls after this request, the RC MUST NOT include the interaction reference. If the AS detects an RC submitting the same interaction reference multiple times, the AS MUST return an error and SHOULD invalidate the ongoing request.

The Section 3 MAY contain any newly-created access tokens (Section 3.2) or newly-released subject claims (Section 3.4). The response MAY contain a new "continue" response (Section 3.1) as described above. The response SHOULD NOT contain any interaction responses (Section 3.3). [[See issue #89 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/89)]]

For example, if the request is successful in causing the AS to issue access tokens and release subject claims, the response could look like this:

```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L"
    },
    "subject": {
        "sub_ids": [ {
           "subject_type": "email",
           "email": "user@example.com",
        } ]
    }
}
```

With this example, the RC can not make an additional continuation request because a "continue" field is not included.

[[See issue #88 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/88)]]

5.2. Continuing During Pending Interaction

When the RC does not include a "callback" parameter, the RC will often need to poll the AS until the RO has authorized the request. To do so, the RC makes a POST request to the continuation URI as in Section 5.1, but does not include a message body.

```
POST /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...
```

The Section 3 MAY contain any newly-created access tokens (Section 3.2) or newly-released subject claims (Section 3.4). The response MAY contain a new "continue" response (Section 3.1) as described above. If a "continue" field is included, it SHOULD include a "wait" field to facilitate a reasonable polling rate by the RC. The response SHOULD NOT contain interaction responses (Section 3.3).

For example, if the request has not yet been authorized by the RO, the AS could respond by telling the RC to make another continuation request in the future. In this example, a new, unique access token has been issued for the call, which the RC will use in its next continuation request.

```
   {
       "continue": {
           "access_token": {
               "value": "33OMUKMKSKU80UPRY5NM",
               "key": true
           },
           "uri": "https://server.example.com/continue",
           "wait": 30
       }
   }
```

[[See issue #90 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/90)]]

[[See issue #91 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/91)]]

If the request is successful in causing the AS to issue access tokens and release subject claims, the response could look like this example:

```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L"
    },
    "subject": {
        "sub_ids": [ {
           "subject_type": "email",
           "email": "user@example.com",
        } ]
    }
}
```

5.3. Modifying an Existing Request

The RC might need to modify an ongoing request, whether or not tokens have already been issued or claims have already been released. In such cases, the RC makes an HTTP PATCH request to the continuation URI and includes any fields it needs to modify. Fields that aren’t included in the request are considered unchanged from the original request.

The RC MAY include the "resources" and "subject" fields as described in Section 2.1 and Section 2.2. Inclusion of these fields override any values in the initial request, which MAY trigger additional requirements and policies by the AS. For example, if the RC is asking for more access, the AS could require additional interaction with the RO to gather additional consent. If the RC is asking for more limited access, the AS could determine that sufficient authorization has been granted to the RC and return the more limited access rights immediately. [[See issue #92 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/92)]]

The RC MAY include the "interact" field as described in Section 2.5. Inclusion of this field indicates that the RC is capable of driving interaction with the RO, and this field replaces any values from a previous request. The AS MAY respond to any of the interaction responses as described in Section 3.3, just like it would to a new request.

The RC MAY include the "user" field as described in Section 2.4 to present new assertions or information about the RQ. [[See issue #93 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/93)]]

The RC MUST NOT include the "client" section of the request. [[See issue #94 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/94)]]

The RC MAY include post-interaction responses such as described in Section 5.1. [[See issue #95 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/95)]]

Modification requests MUST NOT alter previously-issued access tokens. Instead, any access tokens issued from a continuation are considered new, separate access tokens. The AS MAY revoke existing access tokens after a modification has occurred. [[See issue #96 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/96)]]

If the modified request can be granted immediately by the AS, the Section 3 MAY contain any newly-created access tokens (Section 3.2) or newly-released subject claims (Section 3.4). The response MAY contain a new "continue" response (Section 3.1) as described above. If interaction can occur, the response SHOULD contain interaction responses (Section 3.3) as well.

For example, an RC initially requests a set of resources using references:

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

{
    "resources": [
        "read", "write"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.example.net/return/123455",
            "nonce": "LKLTI25DK82FX4T4QFZC"
        }
    },
    "client": "987YHGRT56789IOLK"
}
```

Access is granted by the RO, and a token is issued by the AS. In its final response, the AS includes a "continue" field:

```
{
    "continue": {
        "access_token": {
            "value": "80UPRY5NM33OMUKMKSKU",
            "key": true
        },
        "uri": "https://server.example.com/continue",
        "wait": 30
    },
    "access_token": ...
}
```

This allows the RC to make an eventual continuation call. The RC realizes that it no longer needs "write" access and therefore modifies its ongoing request, here asking for just "read" access instead of both "read" and "write" as before.

```
PATCH /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...

{
    "resources": [
        "read"
    ]
    ...
}
```

The AS replaces the previous "resources" from the first request, allowing the AS to determine if any previously-granted consent already applies. In this case, the AS would likely determine that reducing the breadth of the requested access means that new access tokens can be issued to the RC. The AS would likely revoke previously-issued access tokens that had the greater access rights associated with them.

```
{
    "continue": {
        "access_token": {
            "value": "M33OMUK80UPRY5NMKSKU",
            "key": true
        },
        "uri": "https://server.example.com/continue",
        "wait": 30
    },
    "access_token": ...
}
```

For another example, the RC initially requests read-only access but later needs to step up its access. The initial request could look like this example.

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

{
    "resources": [
        "read"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.example.net/return/123455",
            "nonce": "LKLTI25DK82FX4T4QFZC"
        }
    },
    "client": "987YHGRT56789IOLK"
}
```

Access is granted by the RO, and a token is issued by the AS. In its final response, the AS includes a "continue" field:

```
{
    "continue": {
        "access_token": {
            "value": "80UPRY5NM33OMUKMKSKU",
            "key": true
        },
        "uri": "https://server.example.com/continue",
        "wait": 30
    },
    "access_token": ...
}
```

This allows the RC to make an eventual continuation call. The RC later realizes that it now needs "write" access in addition to the "read" access. Since this is an expansion of what it asked for previously, the RC also includes a new interaction section in case the AS needs to interact with the RO again to gather additional authorization. Note that the RC's nonce and callback are different from the initial request. Since the original callback was already used in the initial exchange, and the callback is intended for one-time-use, a new one needs to be included in order to use the callback again.

[[See issue #97 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/97)]]

```
PATCH /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...

{
    "resources": [
        "read", "write"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.example.net/return/654321",
            "nonce": "K82FX4T4LKLTI25DQFZC"
        }
    }
}
```

From here, the AS can determine that the RC is asking for more than it was previously granted, but since the RC has also provided a mechanism to interact with the RO, the AS can use that to gather the additional consent. The protocol continues as it would with a new request. Since the old access tokens are good for a subset of the rights requested here, the AS might decide to not revoke them. However, any access tokens granted after this update process are new access tokens and do not modify the rights of existing access tokens.

5.4. Getting the Current State of a Grant Request

If the RC needs to get the current state of an ongoing grant request, it makes an HTTP GET request to the continuation URI. This request MUST NOT alter the grant request in any fashion, including causing the issuance of new access tokens or modification of interaction parameters.

The AS MAY include existing access tokens and previously-released subject claims in the response. The AS MUST NOT issue a new access token or release a new subject claim in response to this request.

```
GET /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...
```

The response MAY include any fields described Section 3 that are applicable to this ongoing request, including the most recently issued access tokens, any released subject claims, and any currently active interaction modes. The response MAY contain a new "continue" response (Section 3.1) as described above.

[[See issue #98 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/98)]]

5.5. Canceling a Grant Request

If the RC wishes to cancel an ongoing grant request, it makes an HTTP DELETE request to the continuation URI.

```
DELETE /continue HTTP/1.1
Host: server.example.com
Content-type: application/json
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...
```

If the request is successfully cancelled, the AS responds with an HTTP 202. The AS MUST revoke all associated access tokens, if possible.

6. Token Management

If an access token response includes the "manage" parameter as described in Section 3.2.1, the RC 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 RC 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 RC MUST present proof of the same key identified in the initial request (Section 2.3.2) as described in Section 8.

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

[[See issue #99 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/99)]]

6.1. Rotating the Access Token

The RC 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....
```

[[See issue #100 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/100)]]

The AS validates that the token presented is associated with the management URL, that the AS issued the token to the given RC, and that the presented key is appropriate to the token.

If the access token has expired, the AS SHOULD honor the rotation request to the token management URL since it is likely that the RC is attempting to refresh the expired token. To support this, the AS MAY apply different lifetimes for the use of the token in management vs. its use at an RS. An AS MUST NOT honor a rotation request for an access token that has been revoked, either by the AS or by the RC through the token management URI (Section 6.2).

If the token is validated and the key is appropriate for the request, the AS MUST 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, unless the "multi_token" flag is specified in the request. 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 RC MUST use this new URL to manage the new access token. [[See issue #101 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/101)]]

[[See issue #102 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/102)]]

```
{
    "access_token": {
        "value": "FP6A8H6HY37MH13CK76LBZ6Y1UADG6VEUPEER5H2",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L",
        "resources": [
            {
                "type": "photo-api",
                "actions": [
                    "read",
                    "write",
                    "dolphin"
                ],
                "locations": [
                    "https://server.example.net/",
                    "https://resource.local/other"
                ],
                "datatypes": [
                    "metadata",
                    "images"
                ]
            },
            "read", "dolphin-metadata"
        ]
    }
}
```

[[See issue #103 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/103)]]

6.2. Revoking the Access Token

If the RC wishes to revoke the access token proactively, such as when a user indicates to the RC that they no longer wish for it to have access or the RC application detects that it is being uninstalled, the RC can use the token management URI to indicate to the AS that the AS should invalidate the access token for all purposes.

The RC makes an HTTP DELETE request to the token management URI, presenting the access token and signing the request with the appropriate key.

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

If the key presented is associated with the token (or the RC, in the case of a bearer token), the AS MUST invalidate the access token, if possible, and return an HTTP 204 response code.

```
204 No Content
```

Though the AS MAY revoke an access token at any time for any reason, the token management function is specifically for the RC's use. If the access token has already expired or has been revoked through other means, the AS SHOULD honor the revocation request to the token management URL as valid, since the end result is still the token not being usable.

7. Using Access Tokens

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

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

```
Authorization: Bearer OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0
```

The form parameter and query parameter methods of [RFC6750] MUST NOT be used.

If the "key" value is the boolean "true", the access token MUST be sent to the RS using the same key and proofing mechanism that the RC used in its initial request.

If the "key" value is an object, the value of the "proof" field within the key indicates the particular proofing mechanism to use. 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....
```

[[See issue #104 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/104)]]

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 2.3.2. 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 proofing methods are defined by a registry TBD (Section 12).

All key binding methods used by this specification MUST cover all relevant portions of the request, including anything that would change the nature of the request, to allow for secure validation of the request by the AS. Relevant aspects include the URI being called, the HTTP method being used, any relevant HTTP headers and values, and the HTTP message body itself. The recipient of the signed message MUST validate all components of the signed message to ensure that nothing has been tampered with or substituted in a way that would change the nature of the request.

When used for delegation in GNAP, these key binding mechanisms allow the AS to ensure that the keys presented by the RC in the initial request are in control of the party calling any follow-up or continuation requests. To facilitate this requirement, all keys in the initial request Section 2.3.2 MUST be proved in all continuation requests Section 5 and token management requests Section 6, modulo any rotations on those keys over time that the AS knows about. The AS MUST validate all keys presented by the RC (Section 2.3.2) or referenced in an ongoing request for each call within that request.

[[See issue #105 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/105)]]

When used to bind to an access token, the access token MUST be covered by the signature method.

8.1. Detached JWS

This method is indicated by "jwsd" in the "proof" field. A JWS [RFC7515] signature object is created as follows:

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 "b64" field MUST be set to "false" and the "crit" field MUST contain at least "b64" as specified in [RFC7797]

To protect the request, the JWS header MUST contain the following additional fields.

htm (string) The HTTP Method used to make this request, as an uppercase ASCII string.

htu (string) The HTTP URI used for this request, including all path and query components.

ts (integer) A timestamp of the request in integer seconds

at_hash (string) When to bind a request to an access token, the access token hash value. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the "access_token" value, where the hash algorithm used is the hash algorithm used in the "alg" header parameter of the JWS's JOSE Header. For instance, if the "alg" is "RS256", hash the "access_token" value with SHA-256, then take the left-most 128 bits and base64url encode them.

[[See issue #106 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/106)]]

The payload of the JWS object is the serialized body of the request, and the object is signed according to detached JWS [RFC7797].

The RC presents the signature in the Detached-JWS HTTP Header field. [[See issue #107 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/107)]]

```
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

{
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "client": {
      "proof": "jwsd",
      "key": {
        "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"
        }
      }
      "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
      },
    }
}
```

If the request being made does not have a message body, such as an HTTP GET, OPTIONS, or DELETE method, the JWS signature is calculated over an empty payload.

When the server (AS or RS) 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.

[[See issue #108 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/108)]]

8.2. Attached JWS

This method is indicated by "jws" in the "proof" field. A JWS [RFC7515] signature object is created as follows:

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".

To protect the request, the JWS header MUST contain the following additional fields.

htm (string) The HTTP Method used to make this request, as an uppercase ASCII string.

htu (string) The HTTP URI used for this request, including all path and query components.

ts (integer) A timestamp of the request in integer seconds

at_hash (string) When to bind a request to an access token, the access token hash value. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the "access_token" value, where the hash algorithm used is the hash algorithm used in the "alg" header parameter of the JWS's JOSE Header. For instance, if the "alg" is "RS256", hash the "access_token" value with SHA-256, then take the left-most 128 bits and base64url encode them.

[[See issue #107 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/107)]]

The payload of the JWS object is the JSON serialized body of the request, and the object is signed according to JWS and serialized into compact form [RFC7515].

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 /tx HTTP/1.1
Host: server.example.com
Content-Type: application/jose

eyJhbGciOiJSUzI1NiIsImtpZCI6IktBZ05wV2JSeXk5T
WYycmlrbDQ5OExUaE1ydmtiWldIVlNRT0JDNFZIVTQiLC
JodG0iOiJwb3N0IiwiaHR1IjoiL3R4IiwidHMiOjE2MDM
4MDA3ODN9.eyJjYXBhYmlsaXRpZXMiOltdLCJjbGllbnQ
iOnsia2V5Ijp7Imp3ayI6eyJrdHkiOiJSU0EiLCJlIjoi
QVFBQiIsImtpZCI6IktBZ05wV2JSeXk5TWYycmlrbDQ5O
ExUaE1ydmtiWldIVlNRT0JDNFZIVTQiLCJuIjoibGxXbU
hGOFhBMktOTGRteE9QM2t4RDlPWTc2cDBTcjM3amZoejk
0YTkzeG0yRk5xb1NQY1JaQVBkMGxxRFM4TjNVaWE1M2RC
MjNaNTlPd1k0YnBNX1ZmOEdKdnZwdExXbnhvMVB5aG1Qc
i1lY2RTQ1JRZFRjX1pjTUY0aFJWNDhxcWx2dUQwbXF0Y0
RiSWtTQkR2Y2NKbVpId2ZUcERIaW5UOHR0dmNWUDhWa0F
NQXE0a1ZhenhPcE1vSVJzb3lFcF9lQ2U1cFN3cUhvMGRh
Q1dOS1ItRXBLbTZOaU90ZWRGNE91bXQ4TkxLVFZqZllnR
khlQkRkQ2JyckVUZDR2Qk13RHRBbmpQcjNDVkN3d3gyYk
FRVDZTbHhGSjNmajJoaHlJcHE3cGM4clppYjVqTnlYS3d
mQnVrVFZZWm96a3NodC1Mb2h5QVNhS3BZVHA4THROWi13
In0sInByb29mIjoiandzIn0sIm5hbWUiOiJNeSBGaXN0I
ENsaWVudCIsInVyaSI6Imh0dHA6Ly9sb2NhbGhvc3QvY2
xpZW50L2NsaWVudElEIn0sImludGVyYWN0Ijp7ImNhbGx
iYWNrIjp7Im1ldGhvZCI6InJlZGlyZWN0Iiwibm9uY2Ui
OiJkOTAyMTM4ODRiODQwOTIwNTM4YjVjNTEiLCJ1cmkiO
iJodHRwOi8vbG9jYWxob3N0L2NsaWVudC9yZXF1ZXN0LW
RvbmUifSwicmVkaXJlY3QiOnRydWV9LCJyZXNvdXJjZXM
iOnsiYWN0aW9ucyI6WyJyZWFkIiwicHJpbnQiXSwibG9j
YXRpb25zIjpbImh0dHA6Ly9sb2NhbGhvc3QvcGhvdG9zI
l0sInR5cGUiOiJwaG90by1hcGkifSwic3ViamVjdCI6ey
JzdWJfaWRzIjpbImlzcy1zdWIiLCJlbWFpbCJdfX0.LUy
Z8_fERmxbYARq8kBYMwzcd8GnCAKAlo2ZSYLRRNAYWPrp
2XGLJOvg97WK1idf_LB08OJmLVsCXxCvn9mgaAkYNL_Zj
HcusBvY1mNo0E1sdTEr31CVKfC-6WrZCscb8YqE4Ayhh0
Te8kzSng3OkLdy7xN4xeKuHzpF7yGsM52JZ0cBcTo6WrY
EfGdr08AWQJ59ht72n3jTsmYNy9A6I4Wrvfgj3TNxmwYo
jpBAicfjnzA1UVcNm9F_xiSz1_y2tdH7j5rVqBMQife-k
9Ewk95vr3lurthenliYSNiUinVfoW1ybnaIBcTtP1_YCx
g_h1y-B5uZEvYNGCuoCqa6IQ
```

This example’s JWS header decodes to:

```
   {
     "alg": "RS256",
     "kid": "KAgNpWbRyy9Mf2rikl498LThMrvkbZWHVSQOBC4VHU4",
     "htm": "post",
     "htu": "/tx",
     "ts": 1603800783
   }
```

And the JWS body decodes to:

```
   {
     "capabilities": [],
     "client": {
       "key": {
         "jwk": {
           "kty": "RSA",
           "e": "AQAB",
           "kid": "KAgNpWbRyy9Mf2rikl498LThMrvkbZWHVSQOBC4VHU4",
           "n": "llWmHF8XA2KNLdmxOP3kxD9OY76p0Sr37jfhz94a93xm2FNqoSPc
           RZAPd0lqDS8N3Uia53dB23Z59OwY4bpM_Vf8GJvvptLWnxo1PyhmPr-ecd
           SCRQdTc_ZcMF4hRV48qqlvuD0mqtcDbIkSBDvccJmZHwfTpDHinT8ttvcV
           P8VkAMAq4kVazxOpMoIRsoyEp_eCe5pSwqHo0daCWNKR-EpKm6NiOtedF4
           Oumt8NLKTVjfYgFHeBDdCbrrETd4vBMwDtAnjPr3CVCwwx2bAQT6SlxFJ3
           fj2hhyIpq7pc8rZib5jNyXKwfBukTVYZozksht-LohyASaKpYTp8LtNZ-w"
         },
         "proof": "jws"
       },
       "name": "My Fist Client",
       "uri": "http://localhost/client/clientID"
     },
     "interact": {
       "callback": {
         "method": "redirect",
         "nonce": "d90213884b840920538b5c51",
         "uri": "http://localhost/client/request-done"
       },
       "redirect": true
     },
     "resources": {
       "actions": [
         "read",
         "print"
       ],
       "locations": [
         "http://localhost/photos"
       ],
       "type": "photo-api"
     },
     "subject": {
       "sub_ids": [
         "iss_sub",
         "email"
       ]
     }
   }
```

If the request being made does not have a message body, such as an HTTP GET, OPTIONS, or DELETE method, the JWS signature is calculated over an empty payload and passed in the "Detached-JWS" header as described in Section 8.1.

[[See issue #109 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/109)]]

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 /tx HTTP/1.1
Host: server.example.com
Content-Type: application/json
SSL_CLIENT_CERT: MIIEHDCCAwSgAwIBAgIBATANBgkqhkiG9w0BAQsFADCBmjE3MDUGA1UEAwwuQmVz
 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

{
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "client": {
      "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
      },
      "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"
    }
}
```

[[See issue #110 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/110)]]

8.4. Demonstration of Proof-of-Possession (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. In addition to the required fields, the DPoP body MUST also contain a digest of the request body:

digest (string) Digest of the request body as the value of the Digest header defined in [RFC3230].

```
POST /tx 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

{
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "client": {
      "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
```

```
       },
       "proof": "dpop",
       "key": {
         "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"
         }
       }
     }
 }
```

[[See issue #111 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/111)]]

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] and include this header in the signature.

```
POST /tx 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=

{
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "push",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "client": {
      "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
      },
      "proof": "httpsig",
      "key": {
        "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 Proof of Possession (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 omitted unless this method is being used in conjunction with an access token as in Section 7. [[See issue #112 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/112)]]

* The "b" (body hash) field MUST be calculated and supplied, unless there is no entity body (such as a GET, OPTIONS, or DELETE request).

* All components of the URL MUST be calculated and supplied

* The m (method) field MUST be supplied

```
POST /tx 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

{
    "resources": [
        "dolphin-metadata"
    ],
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.foo",
            "nonce": "VJLO6A4CAYLBXHTR0KRO"
        }
    },
    "client": {
      "display": {
        "name": "My Client Display Name",
        "uri": "https://example.net/client"
      },
      "proof": "oauthpop",
      "key": {
        "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"
        }
      }
    }
}
```

[[See issue #113 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/113)]]

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 grant request endpoint to retrieve the server's discovery information. The AS MUST respond with a JSON document containing the following information:

grant_request_endpoint (string) 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 (array of strings) OPTIONAL. A list of the AS's capabilities. The values of this result MAY be used by the RC in the capabilities section (Section 2.6) of the request.

interaction_methods (array of strings) 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 (array strings) 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 2.3.2) of the request.

sub_ids (array of strings) 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 (array of strings) 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 RC 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 RC 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.

[[See issue #114 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/114)]]

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. [[See issue #115 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/115)]]

```
   +------+       +------+       +------+
   |  RC  |--(1)->|  RS  |       |  AS  |
   |      |       |      |--(2)->|      |
   |      |       |      |<-(3)--|      |
   |      |       |      |       +------+
   |      |<-(4)--|      |
   +------+       +------+
```

1. The RC calls the RS with its access token.

2. The RS introspects the access token value at the AS. The RS signs the request with its own key (not the RC's key or the token's key).

3. The AS validates the token value and the RC's request and returns the introspection response for the token.

4. The RS fulfills the request from the RC.

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"
    ],
    "client": {
      "key": {
        "proof": "httpsig",
        "jwk": {
                    "kty": "RSA",
                    "e": "AQAB",
                    "kid": "xyz-1",
                    "alg": "RS256",
                    "n": "kOB5rR4Jv0GMeL...."
        }
      }
    }
}
```

10.2. Deriving a downstream token

Some architectures require an RS to act as an RC and request a derived access token for a secondary RS. This internal token is issued in the context of the incoming access token.

```
+------+       +-------+       +------+       +-------+
|  RC  |--(1)->|  RS1  |       |  AS  |       |  RS2  |
|      |       |       |--(2)->|      |       |       |
|      |       |       |<-(3)--|      |       |       |
|      |       |       |       +------+       |       |
|      |       |       |                      |       |
|      |       |       |-----------(4)------->|       |
|      |       |       |<----------(5)--------|       |
|      |<-(6)--|       |                      |       |
+------+       +-------+                      +-------+
```

1. The RC calls RS1 with an access token.

2. RS1 presents that token to the AS to get a derived token for use at RS2. RS1 indicates that it has no ability to interact with the RO. RS1 signs its request with its own key, not the token's key or the RC's key.

3. The AS returns a derived token to RS1 for use at RS2.

4. RS1 calls RS2 with the token from (3).

5. RS2 fulfills the call from RS1.

6. RS1 fulfills the call from RC.

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.

[[See issue #116 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/116)]]

```
 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"
     ],
     "client": "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"
    ],
    "client": "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 (Section 10.4) or as documentation to developers.

[[See issue #117 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/117)]]

10.4. Requesting Resources With Insufficient Access

If the RC calls an RS without an access token, or with an invalid access token, the RS MAY respond to the RC with an authentication header indicating that GNAP needs to be used to access the resource. The address of the GNAP endpoint MUST be sent in the "as_uri" parameter. The RS MAY additionally return a resource reference that the RC MAY use in its resource request (Section 2.1). This resource reference handle SHOULD be sufficient for at least the action the RC 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 RC.

```
WWW-Authenticate: GNAP as_uri=http://server.example/tx,resource=FWWIKYBQ6U56NL1
```

The RC 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 RC MAY request additional resources and other information, and MAY request multiple access tokens.

[[See issue #118 (https://github.com/ietf-wg-gnap/gnap-core-protocol/issues/118)]]

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 RC, the AS needs to make sure that it has the permission to do so.

15. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", May 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-06, 4 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-secevent-subject-identifiers-06.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>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[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>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/info/rfc8693>.

[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

* -02

  - Moved all "editor's note" items to GitHub Issues.

* -01

  - "updated_at" subject info timestamp now in ISO 8601 string format.

  - Editorial fixes.

  - Added Aaron and Fabien as document authors.

* -00

  - Initial working group draft.

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"
               ]
           }
       ],
       "client": {
         "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 instance identifier to identify itself in future requests (Section 2.3.1).

```
Content-type: application/json

{
    "interact": {
       "redirect": "https://server.example.com/interact/4CF492MLVMSW9MKMXKHQ",
       "callback": "MBDOFXG4Y5CVJCX821LH"
    }
    "continue": {
        "access_token": {
            "value": "80UPRY5NM33OMUKMKSKU",
            "key": true
        },
        "uri": "https://server.example.com/continue"
    },
    "instance_id": "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_jZQ1R2HZT8BOWYHcLmObM7XHPAdJzT
ZMtKBsaraJ64A
  &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
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...

{
    "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",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L",
        "resources": [{
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        }]
    },
    "continue": {
        "access_token": {
            "value": "80UPRY5NM33OMUKMKSKU",
            "key": true
        },
        "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"
       ],
       "client": "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": {
        "uri": "https://server.example.com/continue/80UPRY5NM33OMUKMKSKU",
        "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. The client signs the request using the same key and method that it did in the first request.

```
POST /continue/80UPRY5NM33OMUKMKSKU HTTP/1.1
Host: server.example.com
Detached-JWS: ejy0...
```

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

Note that the continuation URL 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/BI9QNW6V9W3XFJK4R02D HTTP/1.1
Host: server.example.com
Authorization: GNAP
Detached-JWS: ejy0...
```

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

```
Content-type: application/json

{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L",
        "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"
    ],
    "client": {
      "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",
        "key": true,
        "manage": "https://server.example.com/token",
        "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"
       ],
       "client": "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": {
        "access_token": {
            "value": "80UPRY5NM33OMUKMKSKU",
            "key": true
        },
        "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
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Detached-JWS: ejy0...
```

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": {
        "access_token": {
            "value": "BI9QNW6V9W3XFJK4R02D",
            "key": true
        },
        "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
   Authorization: GNAP BI9QNW6V9W3XFJK4R02D
   Detached-JWS: ejy0...
```

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",
        "key": false,
        "manage": "https://server.example.com/token/PRY5NM33OM4TB8N6BW7OZB8CDFONP
219RP1L",
        "resources": [
            "dolphin-metadata", "some other thing"
        ]
    }
}
```

D.2. Applying OAuth 2 Scopes and Client IDs

While GNAP is not designed to be directly compatible with OAuth 2 [RFC6749], considerations have been made to enable the use of OAuth 2 concepts and constructs more smoothly within GNAP.

In this scenario, the client developer has a "client_id" and set of "scope" values from their OAuth 2 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"
    ],
    "client": "7C7C4AZ9KHRS6X63AJAO",
    "interact": {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "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 pre-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.

Appendix E. JSON Structures and Polymorphism

GNAP makes use of polymorphism within the JSON [RFC8259] structures used for the protocol. Each portion of this protocol is defined in terms of the JSON data type that its values can take, whether it's a string, object, array, boolean, or number. For some fields, different data types offer different descriptive capabilities and are used in different situations for the same field. Each data type provides a different syntax to express the same underlying semantic protocol element, which allows for optimization and simplification in many common cases.

Even though JSON is often used to describe strongly typed structures, JSON on its own is naturally polymorphic. In JSON, the named members of an object have no type associated with them, and any data type can be used as the value for any member. In practice, each member has a semantic type that needs to make sense to the parties creating and consuming the object. Within this protocol, each object member is defined in terms of its semantic content, and this semantic content might have expressions in different concrete data types for different specific purposes. Since each object member has exactly one value in JSON, each data type for an object member field is naturally mutually exclusive with other data types within a single JSON object.

For example, a resource request for a single access token is composed of an array of resource request descriptions while a request for multiple access tokens is composed of an object whose member values are all arrays. Both of these represent requests for access, but the difference in syntax allows the RC and AS to differentiate between the two request types in the same request.

Another form of polymorphism in JSON comes from the fact that the values within JSON arrays need not all be of the same JSON data type. However, within this protocol, each element within the array needs to be of the same kind of semantic element for the collection to make sense, even when the data types are different from each other.

For example, each aspect of a resource request can be described using an object with multiple dimensional components, or the aspect can be requested using a string. In both cases, the resource request is being described in a way that the AS needs to interpret, but with different levels of specificity and complexity for the RC to deal with. An API designer can provide a set of common access scopes as simple strings but still allow RC developers to specify custom access when needed for more complex APIs.

Extensions to this specification can use different data types for defined fields, but each extension needs to not only declare what the data type means, but also provide justification for the data type representing the same basic kind of thing it extends. For example, an extension declaring an "array" representation for a field would need to explain how the array represents something akin to the non-array element that it is replacing.

Authors' Addresses

Justin Richer (editor)
Bespoke Engineering

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

Aaron Parecki
Okta

Email: aaron@parecki.com

URI: https://aaronparecki.com

Fabien Imbault
acert.io

Email: fabien.imbault@acert.io
URI: https://acert.io/