

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 16 February 2021

D. Hardt, Ed.
SignIn.Org
15 August 2020

The Grant Negotiation and Authorization Protocol
draft-hardt-xauth-protocol-14

Abstract

Client software often desires resources or identity claims that are independent of the client. This protocol allows a user and/or resource owner to delegate resource authorization and/or release of identity claims to a server. Client software can then request access to resources and/or identity claims by calling the server. The server acquires consent and authorization from the user and/or resource owner if required, and then returns to the client software the authorization and identity claims that were approved. This protocol may be extended on many dimensions.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 16 February 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

extracted from this document must include Simplified BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	The Grant	4
1.2.	Protocol Roles	4
1.3.	Human Interactions	5
1.4.	Trust Model	7
1.5.	Terminology	8
1.6.	Notational Conventions	10
2.	Exemplar Sequences	11
2.1.	"redirect" Interaction	11
2.2.	"user_code" Interaction	13
2.3.	Independent RO Authorization	14
2.4.	Resource Server Access	15
3.	GS APIs	15
3.1.	GS API Table	16
3.2.	Create Grant	16
3.3.	Verify Grant	19
3.4.	Read Grant	20
3.5.	Request JSON	20
3.5.1.	"client" Object	20
3.5.2.	"interaction" Object	21
3.5.3.	"user" Object	21
3.5.4.	"access" Object	22
3.5.5.	"claims" Object	22
3.6.	Read Access	22
3.7.	GS Options	23
4.	GS Responses	23
4.1.	Grant Response	23
4.2.	Interaction Response	25
4.3.	Wait Response	26
4.4.	Response JSON	26
4.4.1.	"client" Object	27
4.4.2.	"interaction" Object	27
4.4.3.	"access" Object	27
4.4.4.	Access Response Object	27
4.4.5.	"claims" Object	28
4.4.6.	"warnings" JSON Array	28
4.5.	Access JSON	28

4.6.	Response Verification	29
5.	Interaction Modes	29
5.1.	"redirect"	29
5.1.1.	"redirect" verification	30
5.2.	"indirect"	30

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

5.3.	"user_code"	30
6.	RS Access	31
7.	Error Responses	31
8.	Warnings	31
9.	Extensibility	31
10.	Rational	33
11.	Privacy Considerations	35
12.	Security Considerations	35
13.	Acknowledgments	35
14.	IANA Considerations	35
15.	References	35
15.1.	Normative References	35
15.2.	Informative References	37
Appendix A.	Document History	38
A.1.	draft-hardt-xauth-protocol-00	38
A.2.	draft-hardt-xauth-protocol-01	38
A.3.	draft-hardt-xauth-protocol-02	38
A.4.	draft-hardt-xauth-protocol-03	38
A.5.	draft-hardt-xauth-protocol-04	39
A.6.	draft-hardt-xauth-protocol-05	39
A.7.	draft-hardt-xauth-protocol-06	39
A.8.	draft-hardt-xauth-protocol-07	39
A.9.	draft-hardt-xauth-protocol-08	39
A.10.	draft-hardt-xauth-protocol-09	40
A.11.	draft-hardt-xauth-protocol-10	40
A.12.	draft-hardt-xauth-protocol-11	40
A.13.	draft-hardt-xauth-protocol-12	40
A.14.	draft-hardt-xauth-protocol-13	40
A.15.	draft-hardt-xauth-protocol-14	41
Appendix B.	Comparison with OAuth 2.0 and OpenID Connect	41
	Author's Address	42

1. Introduction

EDITOR NOTE

This document captures a number of concepts that may be adopted by the proposed GNAP working group. Please refer to this document as:

XAuth

The use of GNAP in this document is not intended to be a declaration of it being endorsed by the GNAP working group.

This document describes the core Grant Negotiation and Authorization Protocol (GNAP). The protocol supports the widely deployed use cases supported by OAuth 2.0 [[RFC6749](#)] & [[RFC6750](#)], OpenID Connect [[OIDC](#)] - an extension of OAuth 2.0, as well as other extensions. Related

Hardt

Expires 16 February 2021

[Page 3]

Internet-DraftThe Grant Negotiation and Authorization Protocol August 2020

documents include: GNAP - Advanced Features [[GNAP Advanced](#)] and JOSE Authentication [[JOSE Authentication](#)] that describes the JOSE mechanisms for client authentication.

The technology landscape has changed since OAuth 2.0 was initially drafted. More interactions happen on mobile devices than PCs. Modern browsers now directly support asymmetric cryptographic functions. Standards have emerged for signing and encrypting tokens with rich payloads (JOSE) that are widely deployed.

GNAP simplifies the overall architectural model, takes advantage of today's technology landscape, provides support for all the widely deployed use cases, offers numerous extension points, and addresses many of the security issues in OAuth 2.0 by passing parameters securely between parties rather than via a browser redirection.

While GNAP is not backwards compatible with OAuth 2.0, it strives to minimize the migration effort.

The suggested pronunciation of GNAP is "guh-nap".

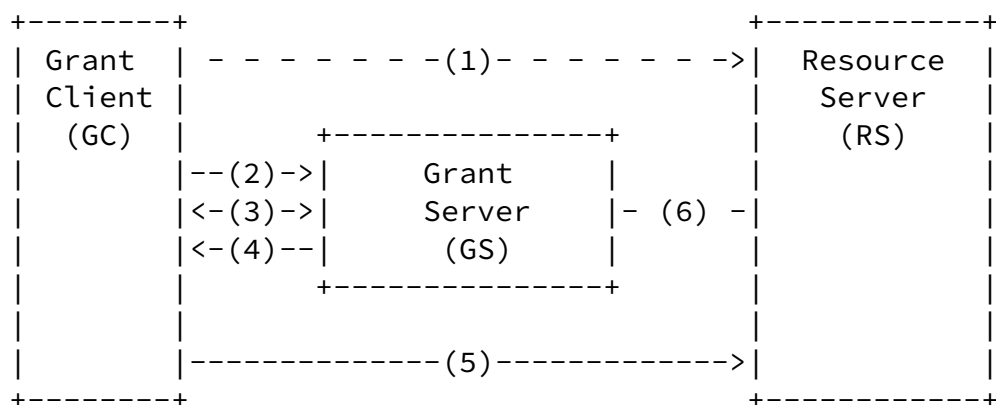
[1.1.](#) The Grant

The Grant is at the center of the protocol between a client and a server. A Grant Client requests a Grant from a Grant Server. The Grant Client and Grant Server negotiate the Grant. The Grant Server acquires authorization to grant the Grant to the Grant Client. The Grant Server then returns the Grant to the Grant Client.

The Grant Request may contain information about the User, the Grant Client, the interaction modes supported by the Grant Client, the requested identity claims, and the requested resource access. Extensions may define additional information to be included in the Grant Request.

1.2. Protocol Roles

There are three roles in GNAP: the Grant Client (GC), the Grant Server (GS), and the Resource Server (RS). Below is how the roles interact:



(1) The GC may query the RS to determine what the RS requires from a GS for resource access. This step is not in scope for this document.

(2) The GC makes a Grant request to the GS (Create Grant [Section 3.2](#)). How the GC authenticates to the GS is not in scope for this document. One mechanism is [\[JOSE Authentication\]](#).

(3) The GC and GS may negotiate the Grant.

(4) The GS returns a Grant to the GC (Grant Response [Section 4.1](#)).

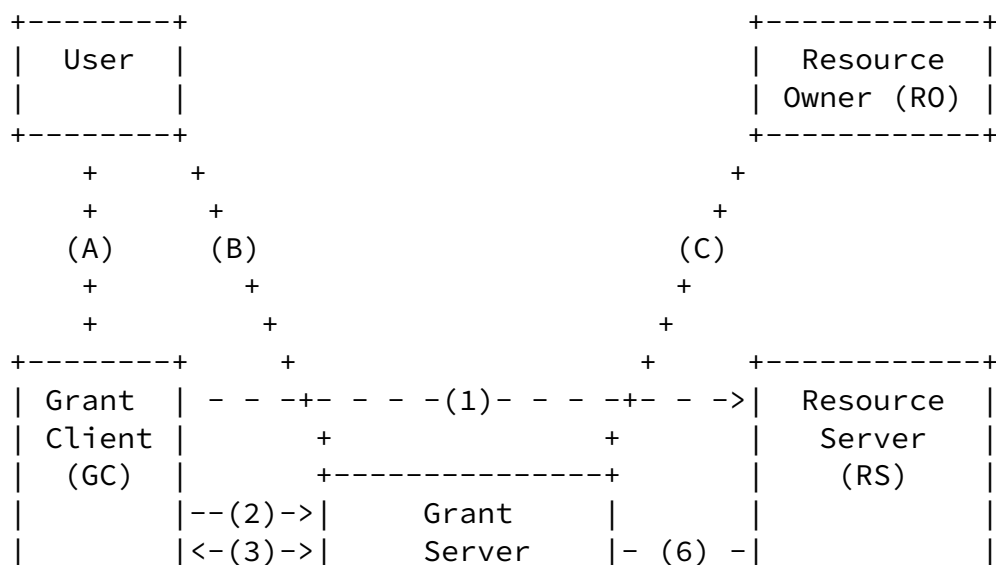
(5) The GC accesses resources at the RS (RS Access [Section 6](#)).

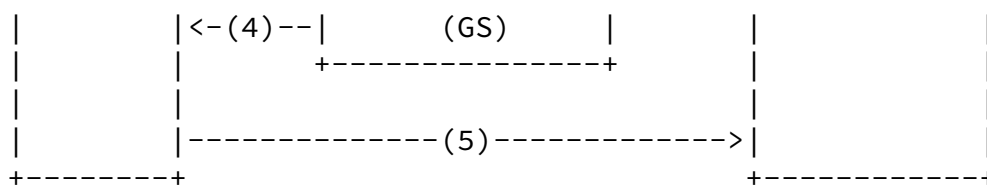
(6) The RS evaluates access granted by the GS to determine access granted to the GC. This step is not in scope for this document.

[1.3.](#) Human Interactions

The Grant Client may be interacting with a human end-user (User), and the Grant Client may need to get authorization to release the Grant from the User, or from the owner of the resources at the Resource Server, the Resource Owner (RO)

Below is when the human interactions may occur in the protocol:





Legend

+ + + indicates an interaction with a human

----- indicates an interaction between protocol roles

Steps (1) - (6) are the same as [Section 1.2](#). The addition of the human interactions (A) - (C) are ***bolded*** below.

(A) The User is interacting with a GC, and the GC needs resource access and/or identity claims (a Grant)

(1) The GC may query the RS to determine what the RS requires from a GS for resource access

(2) The GC makes a Grant request to the GS

(3) The GC and GS may negotiate the Grant

(B) The GS may interact with the User for grant authorization

(C) The GS may interact with the RO for grant authorization

(4) The GS returns a Grant to the GC

(5) The GC accesses resources at the RS

(6) The RS evaluates access granted by the GS to determine access granted to the GC

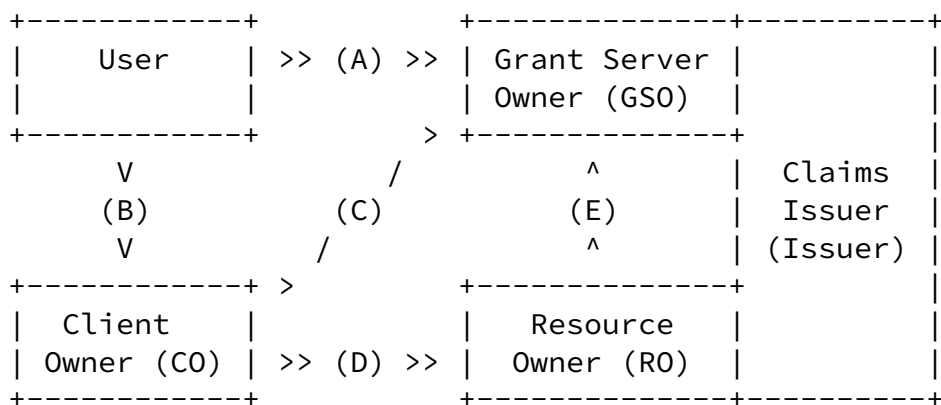
Alternatively, the Resource Owner could be a legal entity that has a software component that the Grant Server interacts with for Grant authorization. This interaction is not in scope of this document.

[1.4.](#) Trust Model

In addition to the User and the Resource Owner, there are three other entities that are part of the trust model:

- * *Client Owner* (CO) - the legal entity that owns the Grant Client.
- * *Grant Server Owner* (GSO) - the legal entity that owns the Grant Server.
- * *Claims Issuer* (Issuer) - a legal entity that issues identity claims about the User. The Grant Server Owner may be an Issuer, and the Resource Owner may be an Issuer.

These three entities do not interact in the protocol, but are trusted by the User and the Resource Owner:



(A) User trusts the GSO to acquire authorization before making a grant to the CO

(B) User trusts the CO to act in the User's best interest with the Grant the GSO grants to the CO

(C) CO trusts claims issued by the GSO

(D) CO trusts claims issued by the RO

(E) RO trusts the GSO to manage access to the RO resources

Roles

* *Grant Client* (GC)

- may want access to resources at a Resource Server
- may be interacting with a User and want identity claims about the User
- requests the Grant Service to grant resource access and identity claims

* *Grant Server* (GS)

- accepts Grant requests from the GC for resource access and identity claims
- negotiates the interaction mode with the GC if interaction is required with the User
- acquires authorization from the User before granting identity claims to the GC
- acquires authorization from the RO before granting resource access to the GC
- grants resource access and identity claims to the GC

* *Resource Server* (RS)

- has resources that the GC may want to access
- expresses what the GC must obtain from the GS for access through documentation or an API. This is not in scope for this document
- verifies the GS granted access to the GC, when the GS makes resource access requests

Humans

* *User*

- the person interacting with the Grant Client.

- has delegated access to identity claims about themselves to the Grant Server.
- may authenticate at the GS.
- * *Resource Owner* (RO)
 - the legal entity that owns resources at the Resource Server (RS).
 - has delegated resource access management to the GS.
 - may be the User, or may be a different entity that the GS interacts with independently.

Reused Terms

- * *access token* - an access token as defined in [\[RFC6749\] Section 1.4](#). An GC uses an access token for resource access at a RS.
- * *Claim* - a Claim as defined in [\[OIDC\] Section 5](#). Claims are issued by a Claims Issuer.
- * *Client ID* - a GS unique identifier for a Registered Client as defined in [\[RFC6749\] Section 2.2](#).
- * *ID Token* - an ID Token as defined in [\[OIDC\] Section 2](#). ID Tokens are issued by the GS. The GC uses an ID Token to authenticate the User.
- * *NumericDate* - a NumericDate as defined in [\[RFC7519\] Section 2](#).
- * *authN* - short for authentication.
- * *authZ* - short for authorization.

New Terms

- * *GS URI* - the endpoint at the GS the GC calls to create a Grant, and is the unique identifier for the GS.
- * *Registered Client* - a GC that has registered with the GS and has a Client ID to identify itself, and can prove it possesses a key that is linked to the Client ID. The GS may have different policies for what different Registered Clients can request. A

Registered Client MAY be interacting with a User.

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * ***Dynamic Client*** - a GC that has not been previously registered with the GS, and each instance will generate its own asymmetric key pair so it can prove it is the same instance of the GC on subsequent requests. The GS MAY return a Dynamic Client a Client Handle for the Dynamic Client to identify itself in subsequent requests. A single-page application with no active server component is an example of a Dynamic Client.
- * ***Client Handle*** - a unique identifier at the GS for a Dynamic Client for the Dynamic Client to refer to itself in subsequent requests.
- * ***Interaction*** - how the GC directs the User to interact with the GS. This document defines the interaction modes: "redirect", "indirect", and "user_code" in [Section 5](#).
- * ***Grant*** - the user identity claims and/or resource access the GS has granted to the Client. The GS MAY invalidate a Grant at any time.
- * ***Grant URI*** - the URI that represents the Grant. The Grant URI MUST start with the GS URI.
- * ***Access*** - the access granted by the RO to the GC and contains an access token. The GS may invalidate an Access at any time.
- * ***Access URI*** - the URI that represents the Access the GC was granted by the RO. The Access URI MUST start with the GS URI. The Access URI is used to refresh an access token.

[1.6](#). Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in [\[RFC2119\]](#).

Certain security-related terms are to be understood in the sense defined in [\[RFC4949\]](#). These terms include, but are not limited to, "attack", "authentication", "authorization", "certificate",

"confidentiality", "credential", "encryption", "identity", "sign", "signature", "trust", "validate", and "verify".

[Editor: review that the terms listed and used are the same]

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

Some protocol parameters are parts of a JSON document, and are referred to in JavaScript notation. For example, "foo.bar" refers to the "bar" boolean attribute in the "foo" object in the following example JSON document:

```
{
  "foo" : {
    "bar": true
  }
}
```

[2.](#) Exemplar Sequences

The following sequences are demonstrative of how GNAP can be used, but are just a few of the possible sequences possible with GNAP.

Before any sequence, the GC needs to be manually or programmatically configured for the GS. See GS Options [Section 3.7](#) for details on programmatically acquiring GS metadata.

In the sequence diagrams:

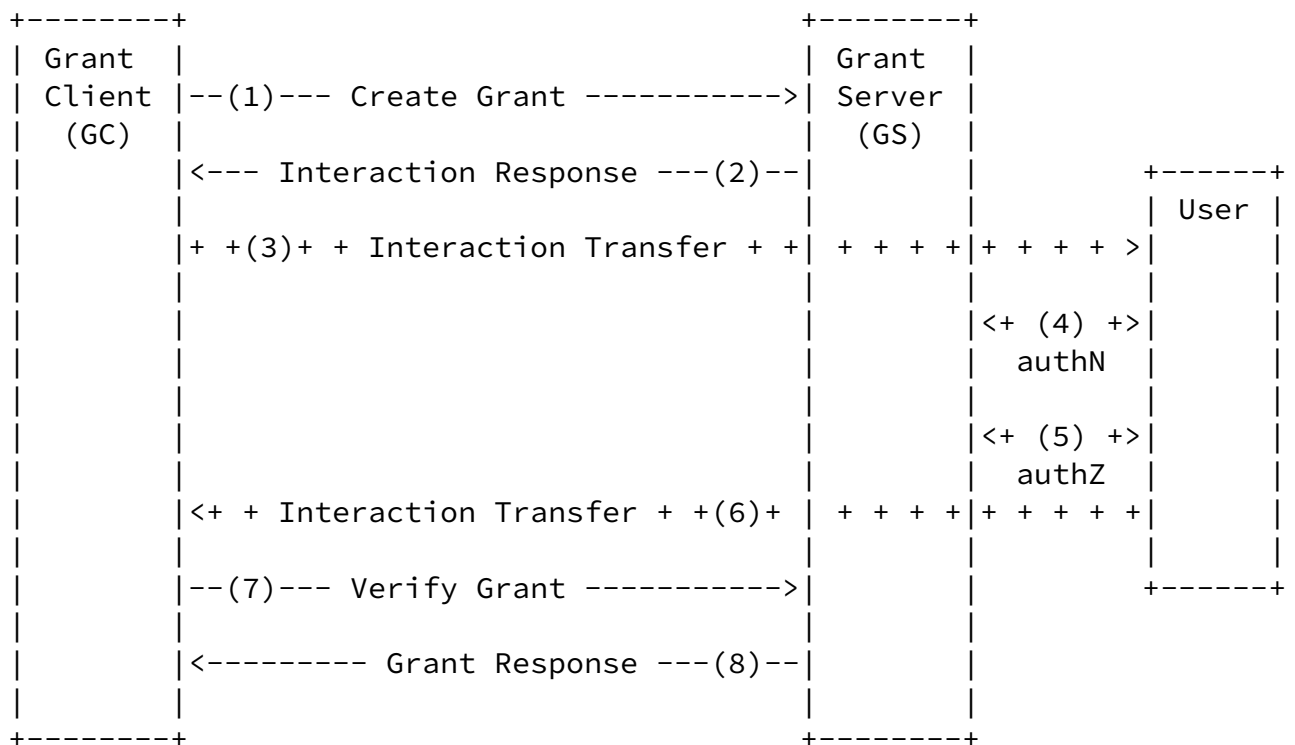
+ + + indicates an interaction with a person

----- indicates an interaction between protocol roles

[2.1.](#) "redirect" Interaction

The GC is a web application and wants a Grant from the User containing resource access and identity claims. The User is the RO for the resource:

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020



1. *Create Grant* The GC creates a Request JSON document [Section 3.5](#) containing an `interaction.redirect` object, and the requested identity claims and resource access. The GC then makes a Create Grant request ([Section 3.2](#)) by sending the JSON with an HTTP POST

to the GS URI.

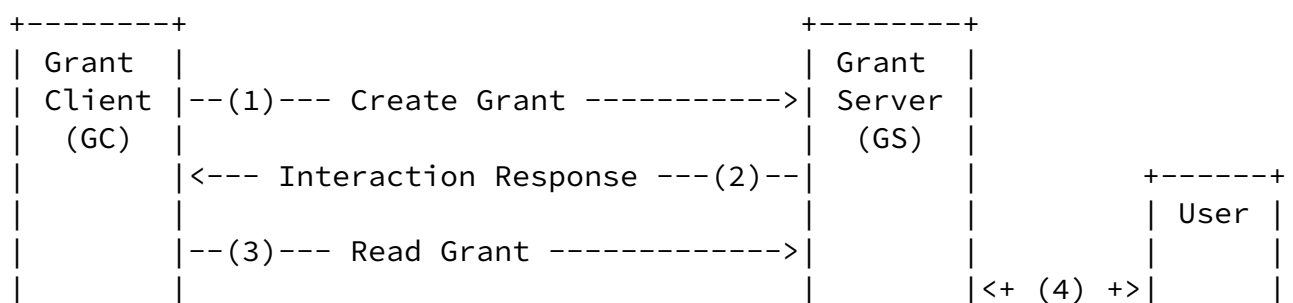
2. *Interaction Response* The GS determines that interaction with the User is required and sends an Interaction Response ([Section 4.2](#)) containing the Grant URI and an interaction.redirect object containing the redirect_uri.
3. *Interaction Transfer* The GC redirects the User to the redirect_uri at the GS.
4. *User Authentication* The GS authenticates the User.
5. *User Authorization* If required, the GS interacts with the User (who may also be the RO) to determine the identity claims and resource access in the Grant Request are to be granted.
6. *Interaction Transfer* The GS redirects the User to the completion_uri at the GC.
7. *Verify Grant* The GC makes an HTTP PATCH request to the Grant URI passing the verification code ([Section 3.3](#)).

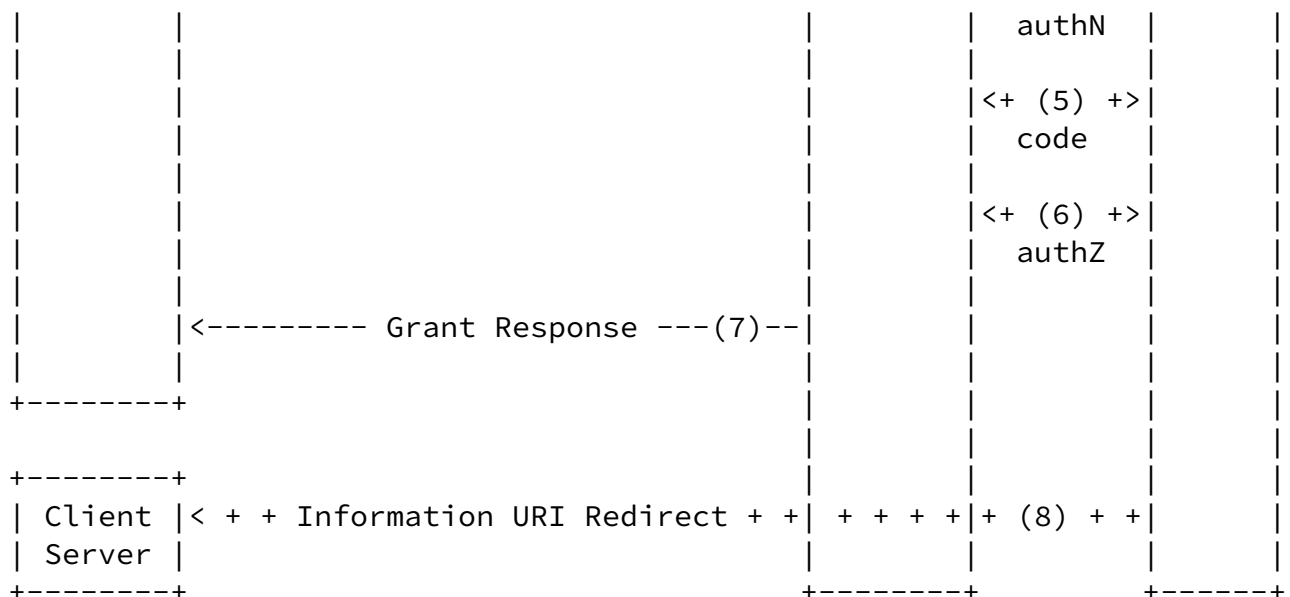
8. *Grant Response* The GS responds with a Grant Response ([Section 4.1](#)).

The GC can now access the resources at the RS per [Section 2.4](#).

[2.2](#). "user_code" Interaction

A GC is on a device that wants a Grant from the User. The User will interact with the GS using a separate device:





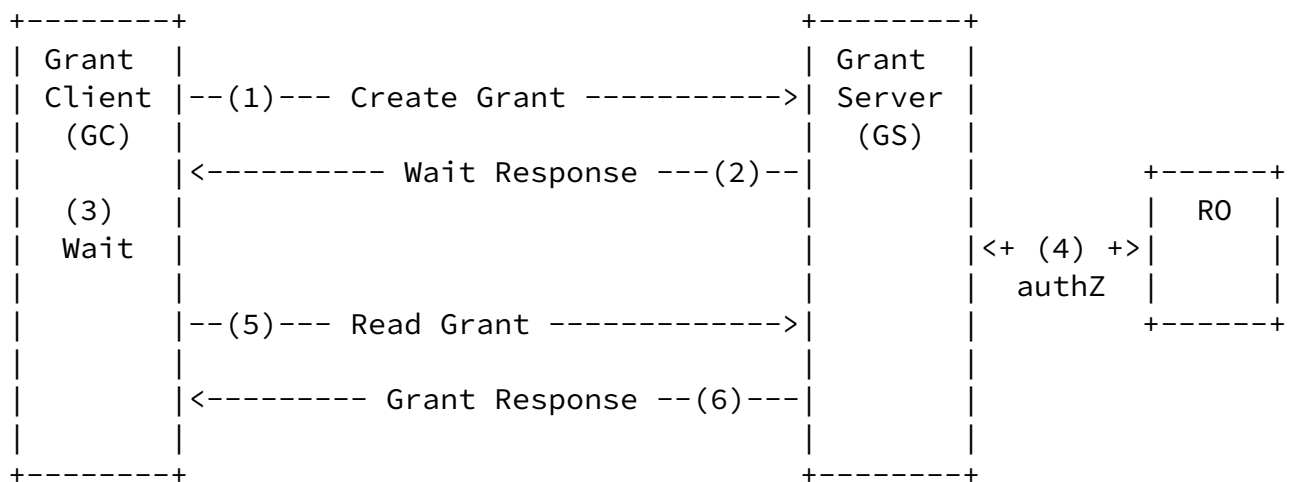
1. ***Create Grant*** The GC creates a Request JSON document ([Section 3.5](#)) containing an `interaction.user_code` object and makes a Create Grant request ([Section 3.2](#)) by sending the JSON with an HTTP POST to the GS URI.
2. ***Interaction Response*** The GS determines that interaction with the User is required and sends an Interaction Response ([Section 4.2](#)) containing the Grant URI and an `interaction.user_code` object.
3. ***Read Grant*** The GC makes an HTTP GET request to the Grant URI.

8. ***Information URI Redirect*** The GS redirects the User to the `information_uri` provided by the GC.

The GC can now access the resources at the RS per [Section 2.4](#).

2.3. Independent RO Authorization

The GC wants access to resources that require the GS to interact with the RO, who is not interacting with the GC. The authorization from the RO may take some time, so the GS instructs the GC to wait and check back later.



1. ***Create Grant*** The GC creates a Grant Request ([Section 3.2](#)) and sends it with an HTTP POST to the GS GS URI.
2. ***Wait Response*** The GS sends an Wait Response ([Section 4.3](#)) containing the Grant URI and the "wait" attribute.
3. ***GC Waits*** The GC waits for the time specified in the "wait" attribute.

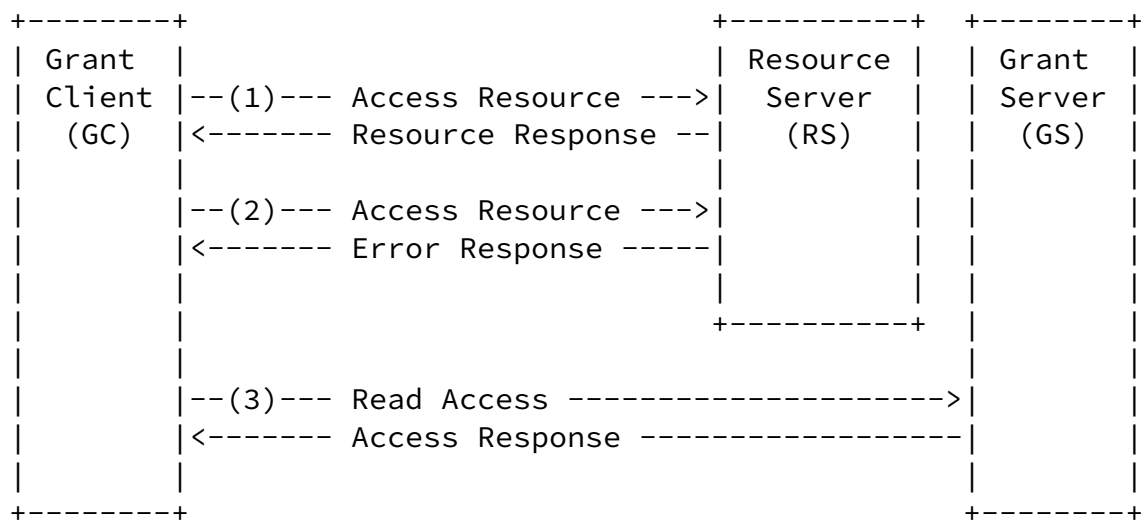
4. ***RO AuthZ*** The GS interacts with the RO to determine which identity claims and/or resource access in the Grant Request are to be granted.

5. ***Read Grant*** The GC does an HTTP GET of the Grant URI ([Section 3.4](#)).
6. ***Grant Response*** The GS responds with a Grant Response ([Section 4.1](#)).

The GC can now access the resources at the RS per [Section 2.4](#).

2.4. Resource Server Access

The GC received an Access URI from the GS. The GC acquires an access token, calls the RS, and later the access token expires. The GC then gets a fresh access token.



1. ***Resource Request*** The GC accesses the RS with the access token per [Section 6](#) and receives a response from the RS.
2. ***Resource Request*** The GC attempts to access the RS, but receives an error indicating the access token needs to be refreshed.
3. ***Read Access*** The GC makes a Read Access ([Section 3.6](#)) with an HTTP GET to the Access URI and receives as Response JSON "access" object ([Section 4.4.4](#)) with a fresh access token.

3. GS APIs

GC Authentication

All GS APIs except for GS Options require the GC to authenticate. Authentication mechanisms include:

- * JOSE Authentication [[JOSE Authentication](#)]
- * [Others TBD]*

[3.1.](#) GS API Table

request	http method	uri	response
Create Grant	POST	GS URI	Interaction, wait, or Grant
Verify Grant	PATCH	Grant URI	Grant
Read Grant	GET	Grant URI	wait, or Grant
Read Access	GET	Access URI	Access
GS Options	OPTIONS	GS URI	metadata

Table 1

[3.2.](#) Create Grant

The GC creates a Grant by doing an HTTP POST of a JSON [[RFC8259](#)] document to the GS URI. This is a Grant Request.

The JSON document MUST include the following from the Request JSON [Section 3.5](#):

- * iat
- * nonce
- * uri - MUST be set to the GS URI
- * method - MUST be "POST"
- * client

and MAY include the following from Request JSON [Section 3.5](#)

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * user
- * interaction
- * access
- * claims

The GS MUST respond with one of Grant Response [Section 4.1](#), Interaction Response [Section 4.2](#), Wait Response [Section 4.3](#), or one of the following errors:

- * TBD

from Error Responses [Section 7](#).

Following is a non-normative example of a web application GC requesting identity claims about the User and read access to the User's contacts:

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

Example 1

```
{
  "iat"      : 15790460234,
  "uri"      : "https://as.example/endpoint",
  "method"   : "POST",
  "nonce"    : "f6a60810-3d07-41ac-81e7-b958c0dd21e4",
  "client": {
    "display": {
      "name"  : "SPA Display Name",
      "uri"   : "https://spa.example/about"
    }
  },
  "interaction": {
    "redirect": {
      "completion_uri" : "https://web.example/return"
    },
    "global" : {
      "ui_locals" : "de"
    }
  },
  "access": [ "read_contacts" ],
  "claims": {
    "oidc": {
      "id_token" : {
        "email"      : { "essential" : true },
        "email_verified" : { "essential" : true }
      },
      "userinfo" : {
        "name"      : { "essential" : true },
        "picture"   : null
      }
    }
  }
}
```

```
}  
}
```

Following is a non-normative example of a device GC requesting two different access tokens, one request with "oauth_scope", the other with "oauth_rich":

Example 2

```
{  
  "iat"      : 15790460234,  
  "uri"      : "https://as.example/endpoint",  
  "method"   : "POST",  
  "nonce"    : "5c9360a5-9065-4f7b-a330-5713909e06c6",  
  "client": {  
    "id"      : "di3872h34dkJW"  
  },  
  "interaction": {  
    "indirect": {  
      "information_uri": "https://device.example/c/indirect"  
    },  
    "user_code": {  
      "information_uri": "https://device.example/c/user_code"  
    }  
  },  
  "access": {  
    "play_music": [ "play_music" ],  
    "read_user_info": [ {  
      "type"      : "customer_information",  
      "locations" : [ "https://example.com/customers" ],  
      "actions"   : [ "read" ],  
      "datatypes" : [ "contacts", "photos" ]  
    } ]  
  }  
}
```

```
}  
}
```

[3.3.](#) Verify Grant

The GC verifies a Grant by doing an HTTP PATCH of a JSON document to the Grant URI. The GC MUST only verify a Grant once.

The JSON document MUST include the following from the Request JSON [Section 3.5](#):

- * iat
- * nonce
- * uri - MUST be set to the Grant URI
- * method - MUST be PATCH
- * interaction.redirection.verifcation - MUST be the verification code received per [Section 5.1.1](#).

Following is a non-normative example:

```
{  
  "iat"      : 15790460235,  
  "uri"      : "https://as.example/endpoint/grant/example1",  
  "method"   : "PATCH",  
  "nonce"    : "9b6afd70-2036-47c9-b953-5dd1fd0c699a",  
  "interaction": {  
    "redirect": {  
      "verification" : "cb4aa22d-2fe1-4321-b87e-bbaa66fbe707"  
    }  
  }  
}
```

The GS MUST respond with one of Grant Response [Section 4.1](#) or one of the following errors:

- * TBD

[3.4.](#) Read Grant

The GC reads a Grant by doing an HTTP GET of the corresponding Grant URI. The GC MAY read a Grant until it expires or has been invalidated.

The GS MUST respond with one of Grant Response [Section 4.1](#), Wait Response [Section 4.3](#), or one of the following errors:

- * TBD

[3.5.](#) Request JSON

- * **iat** - the time of the request as a NumericDate.
- * **nonce** - a unique identifier for this request. Note the Grant Response MUST contain a matching "nonce" attribute value.
- * **uri** - the URI being invoked
- * **method** - the HTTP method being used

[3.5.1.](#) "client" Object

The client object MUST only one of the following:

- * **id** - the Client ID the GS has for a Registered Client.

- * **handle** - the Client Handle the GS previously provided a Dynamic Client
- * **display** - the display object contains the following attributes:
 - **name** - a string that represents the Dynamic Client
 - **uri** - a URI representing the Dynamic Client

The GS will show the the User the display.name and display.uri values when prompting for authorization.

_[Editor: a max length for the name and URI so a GS can reserve appropriate space?]

[3.5.2.](#) "interaction" Object

The interaction object contains one or more interaction mode objects per [Section 5](#) representing the interactions the GC is willing to provide the User. In addition to the interaction mode objects, the interaction object may contain the "global" object;

- * `*global*` - an optional object containing parameters that are applicable for all interaction modes. Only one attribute is defined in this document:
 - `*ui_locales*` - End-User's preferred languages and scripts for the user interface, represented as a space-separated list of [\[RFC5646\]](#) language tag values, ordered by preference. This attribute is OPTIONAL.

_[Editor: ui_locales is taken from OIDC. Why space-separated and not a JSON array?]

[3.5.3.](#) "user" Object

- * `*identifiers*` - The identifiers MAY be used by the GS to improve the User experience. This object contains one or more of the following identifiers for the User:
 - `*phone_number*` - contains a phone number per [Section 5 of \[RFC3966\]](#).
 - `*email*` - contains an email address per [\[RFC5322\]](#).
 - `*oidc*` - is an object containing both the "iss" and "sub" attributes from an OpenID Connect ID Token [\[OIDC\] Section 2](#).

- * `*claims*` - an optional object containing one or more assertions the GC has about the User.
 - `*oidc_id_token*` - an OpenID Connect ID Token per [\[OIDC\] Section 2](#).

[3.5.4.](#) "access" Object

The GC may request a single Access, or multiple. If a single Access, the "access" object contains an array of [\[RAR\]](#) objects. If multiple, the "access" object contains an object where each property name is a unique string created by the GC, and the property value is an array of [\[RAR\]](#) objects.

[3.5.5.](#) "claims" Object

Includes one or more of the following:

- * `*oidc*` - an object that contains one or both of the following objects:
 - `*userinfo*` - Claims that will be returned as a JSON object
 - `*id_token*` - Claims that will be included in the returned ID Token. If the null value, an ID Token will be returned containing no additional Claims.

The contents of the userinfo and id_token objects are Claims as defined in [\[OIDC\] Section 5](#).

- * `*oidc4ia*` - OpenID Connect for Identity Assurance claims request per [\[OIDC4IA\]](#).
- * `*vc*` - `_[Editor: define how W3C Verifiable Credentials can be requested.]_`[\[W3C_VC\]](#)

[3.6.](#) Read Access

The GC acquires and refreshes an Access by doing an HTTP GET to the corresponding Access URI.

The GS MUST respond with a Access JSON document [Section 4.5](#), or one of the following errors:

- * TBD

from Error Responses [Section 7](#).

[3.7.](#) GS Options

The GC can get the metadata for the GS by doing an HTTP OPTIONS of the corresponding GS URI. This is the only API where the GS MAY respond to an unauthenticated request.

The GS MUST respond with the the following JSON document:

- * `*uri*` - the GS URI.
- * `*client_authentication*` - a JSON array of the GC Authentication mechanisms supported by the GS
- * `*interactions*` - a JSON array of the interaction modes supported by the GS.
- * `*access*` - an object containing the access the GC may request from the GS, if any.
 - Details TBD
- * `*claims*` - an object containing the identity claims the GC may request from the GS, if any, and what public keys the claims will be signed with.
 - Details TBD
- * `*algorithms*` - a JSON array of the cryptographic algorithms supported by the GS. [details TBD]*
- * `*features*` - an object containing feature or extension support

or one of the following errors:

- * TBD

from Error Responses [Section 7](#).

[4.](#) GS Responses

There are three successful responses to a Grant Request: Grant Response, Interaction Response, or Wait Response.

[4.1.](#) Grant Response

The Grant Response MUST include the following from the Response JSON [Section 4.4](#)

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * iat
- * nonce
- * uri

and MAY include the following from the Response JSON [Section 4.4](#)

- * client.handle
- * access
- * claims
- * expires_in
- * warnings

Example non-normative Grant Response JSON document for Example 1 in [Section 3.2](#):

```
{
  "iat"          : 15790460234,
  "nonce"        : "f6a60810-3d07-41ac-81e7-b958c0dd21e4",
  "uri"          : "https://as.example/endpoint/grant/example1",
  "expires_in"   : 300
  "access": {
    "mechanism"   : "bearer",
    "token"       : "eyJJJ2D6.example.access.token.mZf9p"
    "expires_in"  : 3600,
    "granted"     : [ "read_contacts" ],
  },
  "claims": {
    "oidc": {
      "id_token"   : "eyJhbUZI1N.example.id.token.YRw5DFdbW",
      "userinfo" : {
        "name"     : "John Doe",
        "picture"  : "https://photos.example/p/eyJzdki0"
      }
    }
  }
}
```

Note in this example since no Access URI was returned in the access object, the access token can not be refreshed, and expires in an hour.

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

Example non-normative Grant Response JSON document for Example 2 in [Section 3.2](#):

```
{
  "iat"    : 15790460234,
  "nonce"  : "5c9360a5-9065-4f7b-a330-5713909e06c6",
  "uri"    : "https://as.example/endpoint/grant/example2",
  "access": {
    "play_music": { "uri"      : "https://as.example/endpoint/access/example2",
                     "read_user_info": { "uri"      : "https://as.example/endpoint/access/" }
  }
}
```

Note in this example the GS only provided the Access URIs. The GC must acquire the Access per [Section 3.6](#)

[Editor: the GC needs to remember if it asked for a single access, or multiple, as there is no crisp algorithm for differentiating between the responses]

[4.2](#). Interaction Response

The Interaction Response MUST include the following from the Response JSON [Section 4.4](#)

- * iat
- * nonce
- * uri
- * interaction

and MAY include the following from the Response JSON [Section 4.4](#)

- * user
- * wait
- * warnings

A non-normative example of an Interaction Response follows:

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

```
{
  "iat"       : 15790460234,
  "nonce"     : "0d1998d8-fbfa-4879-b942-85a88bff1f3b",
  "uri"       : "https://as.example/endpoint/grant/example4",
  "interaction" : {
    "redirect" : {
      "redirect_uri" : "https://as.example/i/example4"
    }
  }
}
```

[4.3.](#) Wait Response

The Wait Response MUST include the following from the Response JSON [Section 4.4](#)

- * iat
- * nonce
- * uri
- * wait

and MAY include the following from the Response JSON [Section 4.4](#)

- * warnings

A non-normative example of Wait Response follows:

```
{
  "iat"      : 15790460234,
  "nonce"    : "0d1998d8-fbfa-4879-b942-85a88bff1f3b",
  "uri"      : "https://as.example/endpoint/grant/example5",
  "wait"     : 300
}
```

[4.4.](#) Response JSON

Details of the JSON document:

- * **iat** - the time of the response as a NumericDate.
- * **nonce** - the nonce that was included in the Request JSON [Section 3.5](#).
- * **uri** - the Grant URI.

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * **wait** - a numeric value representing the number of seconds the GC should wait before making a Read Grant request to the Grant URI.
- * **expires_in** - a numeric value specifying how many seconds until the Grant expires. This attribute is OPTIONAL.

[4.4.1.](#) "client" Object

If the GC is a Dynamic Client, the GS may return

- * **handle** the Client Handle

[4.4.2.](#) "interaction" Object

If the GS wants the GC to start the interaction, the GS MUST return an interaction object containing one or more interaction mode responses per [Section 5](#) to one or more of the interaction mode requests provided by the GC.

[4.4.3.](#) "access" Object

If the GC requested a single Access, the "access" object is an access response object [Section 4.4.4](#). If the GC requested multiple, the access object contains a property of the same name for each Access requested by the GC, and each property is an access response object [Section 4.4.4](#).

[4.4.4](#). Access Response Object

The access response object contains properties from the Access JSON [Section 4.5](#). The access response object MUST contain either the "uri" property from, or MUST contain:

- * mechanism

- * token

and MAY contain:

- * access

- * expires_in

- * uri

If there is no "uri" property, the access token can not be refreshed. If only the "uri" property is present, the GC MUST acquire the Access per [Section 3.6](#)

[4.4.5](#). "claims" Object

The claims object is a response to the Grant Request "claims" object [Section 3.5.5](#).

- * *oidc*

- *id_token* - an OpenID Connect ID Token containing the Claims the User consented to be released.

- *userinfo* - the Claims the User consented to be released.

Claims are defined in [[OIDC](#)] [Section 5](#).

- * `*oidc4ia*` - OpenID Connect for Identity Assurance claims response per [\[OIDC4IA\]](#).
- * `*vc*`

The verified claims the user consented to be released. `_[Editor: details TBD]_`

[4.4.6.](#) "warnings" JSON Array

Includes zero or more warnings from [Section 8](#),

[4.5.](#) Access JSON

The Access JSON is a Grant Response Access Object [Section 4.4.4](#) or the response to a Read Access request by the GC [Section 3.6](#).

- * `*mechanism*` - the RS access mechanism. This document defines the "bearer" mechanism as defined in [Section 6](#). Required.
- * `*token*` - the access token for accessing an RS. Required.
- * `*expires_in*` - an optional numeric value specifying how many seconds until the access token expires.
- * `*uri*` - the Access URI. Used to acquire or refresh Access. Required.
- * `*granted*` - an optional array of [\[RAR\]](#) objects containing the resource access granted

`_[Editor: would an optional expiry for the Access be useful?]_`

The following is a non-normative example of Access JSON:

```
{
  "mechanism"      : "bearer",
  "token"          : "eyJJJ2D6.example.access.token.mZf9p"
  "expires_in"     : 3600,
  "uri"            : "https://as.example/endpoint/access/example2",
  "granted"        : [ "read_calendar write_calendar" ]
}
```


[4.6.](#) Response Verification

On receipt of a response, the GC MUST verify the following:

- * TBD

[5.](#) Interaction Modes

This document defines three interaction modes: "redirect", "indirect", and "user_code". Extensions may define additional interaction modes.

The "global" attribute is reserved in the interaction object for attributes that apply to all interaction modes.

[5.1.](#) "redirect"

A Redirect Interaction is characterized by the GC redirecting the User's browser to the GS, the GS interacting with the User, and then GS redirecting the User's browser back to the GC. The GS correlates the Grant Request with the unique `redirect_uri`, and the GC correlates the Grant Request with the unique `completion_uri`.

The request "interaction" object contains:

- * `*completion_uri*` a unique URI at the GC that the GS will return the User to. The URI MUST not contain the "nonce" from the Grant Request, and MUST not be guessable. This attribute is REQUIRED.

The response "interaction" object contains:

- * `*redirect_uri*` a unique URI at the GS that the GC will redirect the User to. The URI MUST not contain the "nonce" from the Grant Request, and MUST not be guessable. This attribute is REQUIRED.
- * `*verification*` a boolean value indicating the GS requires the GC to make a Verify Grant request. ([Section 3.3](#))

[5.1.1.](#) "redirect" verification

If the GS indicates that Grant Verification is required, the GS MUST add a 'verification' query parameter with a value of a unique verification code to the completion_uri.

On receiving the verification code in the redirect from the GS, the GC makes a Verify Grant request ([Section 3.3](#)) with the verification code.

[5.2.](#) "indirect"

An Indirect Interaction is characterized by the GC causing the User's browser to load the indirect_uri at GS, the GS interacting with the User, and then the GS MAY optionally redirect the User's Browser to a information_uri. There is no mechanism for the GS to redirect the User's browser back to the GC.

Examples of how the GC may initiate the interaction are encoding the indirect_uri as a code scannable by the User's mobile device, or launching a system browser from a command line interface (CLI) application.

The "indirect" mode is susceptible to session fixation attacks. See TBD in the Security Considerations for details.

The request "interaction" object contains:

- * *information_uri* an OPTIONAL URI that the GS will redirect the User's browser to after GS interaction.

The response "interaction" object contains:

- * *indirect_uri* the URI the GC will cause to load in the User's browser. The URI SHOULD be short enough to be easily encoded in a scannable code. The URI MUST not contain the "nonce" from the Grant Request, and MUST not be guessable. _[Editor: recommend a maximum length?]

[5.3.](#) "user_code"

An Indirect Interaction is characterized by the GC displaying a code and a URI for the User to load in a browser and then enter the code. _[Editor: recommend a minimum entropy?]

The request "interaction" object contains:

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * `*information_uri*` an OPTIONAL URI that the GS will redirect the User's browser to after GS interaction.

- *The response "interaction" object contains:

- * `*code*` the code the GC displays to the User to enter at the `display_uri`. This attribute is REQUIRED.

- * `*display_uri*` the URI the GC displays to the User to load in a browser to enter the code.

[6.](#) RS Access

The mechanism the GC MUST use to access an RS is in the Access JSON "mechanism" attribute [Section 4.4.4](#).

The "bearer" mechanism is defined in [Section 2.1 of \[RFC6750\]](#)

The "jose" and "jose+body" mechanisms are defined in [\[JOSE Authentication\]](#)

A non-normative "bearer" example of the HTTP request headers follows:

```
GET /calendar HTTP/2
Host: calendar.example
Authorization: bearer eyJJ2D6.example.access.token.mZf9pTSpA
```

[7.](#) Error Responses

- * TBD

[8.](#) Warnings

[Editor: Warnings are an optional response that can assist a GC in detecting non-fatal errors, such as ignored objects and properties.]

- * TBD

[9.](#) Extensibility

This standard can be extended in a number of areas:

- * `*GC Authentication Mechanisms*`

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- An extension could define other mechanisms for the GC to authenticate to the GS and/or RS such as Mutual TLS or HTTP Signing. Constrained environments could use CBOR [[RFC7049](#)] instead of JSON, and COSE [[RFC8152](#)] instead of JOSE, and CoAP [[RFC8323](#)] instead of HTTP/2.
- * *Grant*
- An extension can define new objects in the Grant Request and Grant Response JSON that return new URIs.
- * *Top Level*
- Top level objects SHOULD only be defined to represent functionality other than the existing top level objects and attributes.
- * *"client" Object*
- Additional information about the GC that the GS would require related to an extension.
- * *"user" Object*
- Additional information about the User that the GS would require related to an extension.
- * *"access" Object*
- RAR is inherently extensible.
- * *"claims" Object*
- Additional claim schemas in addition to OpenID Connect claims and Verified Credentials.

* *interaction modes*

- Additional types of interactions a GC can start with the User.

* *Continuous Authentication*

- An extension could define a mechanism for the GC to regularly provide continuous authentication signals and receive responses.

[Editor: do we specify access token introspection in this document, or leave that to an extension?]

[10.](#) Rational

1. *Why do GCs now always use Asymmetric cryptography? Why not keep the client secret?*

In the past, asymmetric cryptography was relatively computationally expensive. Modern browsers now have asymmetric cryptographic APIs available, and modern hardware has significantly reduced the computational impact.

2. *Why have both Client ID and Client Handle?*

While they both refer to a Grant Client in the protocol, the Client ID refers to a pre-registered client, and the Client Handle is specific to an instance of a Dynamic Client. Using separate terms clearly differentiates which identifier is being presented to the GS.

3. *Why allow GC and GS to negotiate the user interaction mode?*

The GC knows what interaction modes it is capable of, and the GS knows which interaction modes it will permit for a given Grant Request. The GC can then present the intersection to the User to choose which one is preferred. For example, while a device based GC may be willing to do both "indirect" and "user_code", a GS may not enable "indirect" for concern of a session fixation attack. Additional interaction modes will likely become available which allows new modes to be negotiated between GC and GS as each adds additional interaction modes.

4. *Why have both identity claims and resource access?*

There are use cases for each that are independent: authenticating a user and providing claims vs granting access to a resource. A request for an authorization returns an access token which may have full CRUD capabilities, while a request for a claim returns the claim about the User - with no create, update or delete capabilities. While the UserInfo endpoint in OIDC may be thought of as a resource, separating the concepts and how they are requested keeps each of them simpler in the Editor's opinion. :)

5. *Why do some of the JSON objects only have one child, such as the identifiers object in the user object in the Grant Request?*

It is difficult to forecast future use cases. Having more resolution may mean the difference between a simple extension, and a convoluted extension. For example, the "global" object in the "interaction" object allows new global parameters to be added without impacting new interaction modes.

6. *Why is the "iss" included in the "oidc" identifier object? Would the "sub" not be enough for the GS to identify the User?*

This decouples the GS from the OpenID Provider (OP). The GS identifier is the GS URI, which is the endpoint at the GS. The OP issuer identifier will likely not be the same as the GS URI. The GS may also provide claims from multiple OPs.

7. *Why is there not a UserInfo endpoint as there is with OpenID Connect?*

Since the GC can Read Grant at any time, it get the same functionality as the UserInfo endpoint, without the GC having to manage a separate access token and refresh token. If the GC would like additional claims, it can Update Grant, and the GS will let the GC know if an interaction is required to get any of

the additional claims, which the GC can then start.

[Editor: is there some other reason to have the UserInfo endpoint?]

8. *Why use URIs for the Grant and Access?*
 - * Grant URI and Access URI are defined to start with the GS URI, allowing the GC, and GS to determine which GS a Grant or Access belongs to.
 - * URIs also enable a RESTful interface to the GS functionality.
 - * A large scale GS can easily separate out the services that provide functionality as routing of requests can be done at the HTTP layer based on URI and HTTP method. This allows a separation of concerns, independent deployment, and resiliency.
9. *Why use the OPTIONS method on the GS URI? Why not use a .well-known mechanism?*

Having the GS URI endpoint respond to the metadata allows the GS to provide GC specific results using the same GC authentication used for other requests to the GS. It also reduces the risk of a mismatch between the advertised metadata, and the actual metadata. A .well-known discovery mechanism may be defined to resolve from a hostname to the GS URI.

10. *Why is there a Verify Grant? The GC can protect itself from session fixation without it.*

GC implementations may not always follow the best practices. The Verify Grant allows the GS to ensure there is not a session fixation as the instance of the GC making creating the Grant is the one that gets the verification code in the redirect.

11. **Why use the [OIDC] claims rather than the [IANA_JWT] list of claims?

The [IANA_JWT] claims include claims that are not identity claims, and [IANA_JWT] references the [OIDC] claims, and [OIDC] 5.1 are only identity claims.

11. Privacy Considerations

TBD

12. Security Considerations

TBD

13. Acknowledgments

This draft derives many of its concepts from Justin Richer's Transactional Authorization draft [TxAuth].

Additional thanks to Justin Richer and Annabelle Richard Backman for their strong critique of earlier drafts. [Editor: add in the other contributors from mail list]

14. IANA Considerations

TBD

15. References

15.1. Normative References

[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>>.

[RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", [RFC 3966](#), DOI 10.17487/RFC3966, December 2004, <<https://www.rfc-editor.org/info/rfc3966>>.

- [RFC5322] Resnick, P., Ed., "Internet Message Format", [RFC 5322](#), DOI 10.17487/RFC5322, October 2008, <<https://www.rfc-editor.org/info/rfc5322>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, [RFC 4949](#), DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [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>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", [RFC 7519](#), DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [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>>.
- [OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", 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>.

Rich Authorization Requests", January 2020,
<<https://tools.ietf.org/html/draft-ietf-oauth-rar-00>>.

[W3C_VC] Sporny, M., Noble, G., and D. Chadwick, "Verifiable Credentials Data Model 1.0", November 2019,
<<https://w3c.github.io/vc-data-model/>>.

[JOSE_Authentication] Hardt, D., "JOSE Authentication", June 2020,
<<https://tools.ietf.org/html/draft-hardt-gnap-jose>>.

[GNAP_Advanced] Hardt, D., "The Grant Negotiation and Authorization Protocol – Advanced Features", June 2020,
<<https://tools.ietf.org/html/draft-hardt-gnap-advanced>>.

15.2. Informative References

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", [RFC 7049](#), DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", [RFC 8152](#), DOI 10.17487/RFC8152, July 2017,
<<https://www.rfc-editor.org/info/rfc8152>>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", [RFC 8323](#), DOI 10.17487/RFC8323, February 2018,
<<https://www.rfc-editor.org/info/rfc8323>>.

[browser_based_apps] Parecki, A. and D. Waite, "OAuth 2.0 for Browser-Based Apps", September 2019, <<https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps-04>>.

[QR_Code] "ISO/IEC 18004:2015 – Information technology – Automatic identification and data capture techniques – QR Code bar code symbology specification", February 2015,
<<https://www.iso.org/standard/62021.html>>.

[TxAuth] Richer, J., "Transactional AuthN", December 2019,
<<https://tools.ietf.org/html/draft-richer-transactional-authz-04>>.

[IANA_JWT] "JSON Web Token Claims", January 2015,
<<https://www.iana.org/assignments/jwt/jwt.xhtml>>.

Appendix A. Document History

A.1. draft-hardt-xauth-protocol-00

- * Initial version

A.2. draft-hardt-xauth-protocol-01

- * text clean up
- * added OIDC4IA claims
- * added "jws" method for accessing a resource.
- * renamed Initiation Request -> Grant Request
- * renamed Initiation Response -> Interaction Response
- * renamed Completion Request -> Authorization Request
- * renamed Completion Response -> Grant Request
- * renamed completion handle -> authorization handle
- * added Authentication Request, Authentication Response, authentication handle

A.3. draft-hardt-xauth-protocol-02

- * major rewrite
- * handles are now URIs
- * the collection of claims and authorizations are a Grant
- * an Authorization is its own type
- * lots of sequences added

A.4. draft-hardt-xauth-protocol-03

- * fixed R0 definition

- * improved language in Rationals

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * added user code interaction method, and aligned qrcode interaction method
- * added information_uri for code flows

[A.5. draft-hardt-xauth-protocol-04](#)

- * renamed interaction uris to have purpose specific names

[A.6. draft-hardt-xauth-protocol-05](#)

- * separated claims from identifiers in request user object
- * simplified reciprocal grant flow
- * reduced interactions to redirect and indirect
- * simplified interaction parameters
- * added in language for Client to verify interaction completion
- * added Verify Grant API and Interaction Nonce
- * replaced Refresh AuthZ with Read AuthZ. Read and refresh are same operation.

[A.7. draft-hardt-xauth-protocol-06](#)

- * fixup examples to match specification

[A.8. draft-hardt-xauth-protocol-07](#)

- * refactored interaction request and response syntax, and enabled interaction mode negotiation
- * generation of client handle by GS for dynamic clients
- * renamed title to Grant Negotiation and Authorization Protocol. Preserved [draft-hardt-xauth-protocol](#) filename to ease tracking

changes.

- * changed Authorizations to be key / value pairs (aka dictionary) instead of a JSON array

[A.9. draft-hardt-xauth-protocol-08](#)

- * split document into three documents: core, advanced, and JOSE authentication.

Hardt

Expires 16 February 2021

[Page 39]

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * grouped access granted into "access" object in Authorization JSON
- * added warnings object to the Grant Response JSON

[A.10. draft-hardt-xauth-protocol-09](#)

- * added editorial note that this document should be referred to as XAuth

[A.11. draft-hardt-xauth-protocol-10](#)

- * added example of RAR authorization request
- * fixed typos

[A.12. draft-hardt-xauth-protocol-11](#)

- * renamed authorization_uri to interaction_uri to avoid confusion with AZ URI
- * made URI names more consistent
 - renamed completion_uri to information_uri
 - renamed redirect_uri to completion_uri
 - renamed interaction_uri to redirect_uri
 - renamed short_uri to indirect_uri
- * editorial fixes

- * renamed http verb to method
- * added Verify Grant and verification parameters

[A.13. draft-hardt-xauth-protocol-12](#)

- * removed authorization object, and made authorizations object polymorphic

[A.14. draft-hardt-xauth-protocol-13](#)

- * added Q about referencing OIDC claims vs IANA JWT
- * made all authorizations be a RAR type as it provides the required flexibility, removed "oauth_rar" type

Hardt

Expires 16 February 2021

[Page 40]

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * added R0 to places where the R0 and User are the same

[A.15. draft-hardt-xauth-protocol-14](#)

- * rewrote introduction
- * add in claims issuer and grant server owner
- * abstract protocol
- * add clarification on different parties
- * renamed Client to Grant Client
- * added entity relationship diagram
- * updated diagrams
- * added placeholder for Privacy Considerations
- * renamed Authorization to Access

[Appendix B. Comparison with OAuth 2.0 and OpenID Connect](#)

Changed Features

The major changes between GNAP and OAuth 2.X and OpenID Connect are:

- * The OAuth 2.X client and the OpenID Connect replying party are the Grant Client in GNAP.
- * The GNAP Grant Server is a superset of the OAuth 2.X authorization server, and the OpenID Connect OP (OpenID Provider).
- * The GC always uses a private asymmetric key to authenticate to the GS. There is no client secret.
- * The GC initiates the protocol by making a signed request directly to the GS instead of redirecting the User to the GS.
- * The GC does not pass any parameters in redirecting the User to the GS.
- * The refresh_token has been replaced with an AZ URI that both represents the authorization, and is the URI for obtaining a fresh access token.

Internet-Draft The Grant Negotiation and Authorization Protocol August 2020

- * The GC can request identity claims to be returned independent of the ID Token.
- * The GS URI is the only static endpoint. All other URIs are dynamically generated. The GC does not need to register its redirect URIs.

TBD - negotiation

Preserved Features

- * GNAP reuses the scopes, Client IDs, and access tokens of OAuth 2.0.
- * GNAP reuses the Client IDs, Claims and ID Token of OpenID Connect.
- * No change is required by the GC or the RS for accessing existing bearer token protected APIs.

New Features

- * All GC calls to the GS are authenticated with asymmetric cryptography
- * A Grant represents both the user identity claims and RS access granted to the GC.
- * Support for scannable code initiated interactions.
- * Highly extensible per [Section 9](#).

Author's Address

Dick Hardt (editor)
SignIn.Org
United States

Email: dick.hardt@gmail.com