Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2020

J. Richer, Ed.
Bespoke Engineering

July 3, 2019

Transactional Authorization
draft-richer-transactional-authz-01

Abstract

This document defines a mechanism for delegating authorization to a piece of software, and conveying that delegation to the software.

Requirements Language

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 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on January 4, 2020.

Copyright Notice

Table of Contents

1. Protocol 3
1.1. Parties 3
1.2. Sequence 3
2. Transaction request 3
2.1. Client 4
2.2. Resource 5
2.3. User 5
2.4. Interact 6
2.5. Keys 6
3. Interaction response 7
3.1. Redirect interaction 8
3.2. Callback response 9
3.3. Secondary device interaction 9
4. Wait response 10
5. Interaction at the AS 11
6. Error response 11
7. Transaction continue request 12
8. Token response 13
8.1. Presenting Tokens to the RS 13
9. Handle references 13
9.1. Presenting handles 14
9.2. Validating handles 14
9.3. Transaction handles 14
9.4. Client handles 15
9.5. Resource handles 15
9.5.1. Resource-first 16
9.6. User handles 17
9.7. Key handles 17
10. Binding Keys 18
10.1. Binding a key to a transaction 18
10.2. Detached JWS 19
10.3. Validating MTLS 19
10.4. Validating DID 19
11. Acknowledgements 19
12. IANA Considerations 19
13. Security Considerations 19
14. Privacy Considerations 20
15. Normative References 20
Appendix A. Document History 21
Author's Address 21

1. Protocol

This protocol allows a piece of software to request delegated authorization to an API, protected by an authorization server usually on behalf of a resource owner.

1.1. Parties

The Authorization Server (AS) manages the transactions. It is defined by its transaction endpoint, a single URL that accepts a POST request with a JSON payload. The AS MAY also have other endpoints, including interaction endpoints, but these are

The Resource Client (RC) requests tokens from the AS and uses tokens at the RS.

The Resource Server (RS) accepts tokens from the RC and validates them (potentially at the AS).

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

1.2. Sequence

1. The RC creates a transaction request and sends it to the AS

2. The AS processes the transaction request and determines if the RO needs to interact

3. If interaction is required, the AS interacts with the RO, possibly by directing the RC to send the RO there

4. The RC continues the transaction at the AS

5. The AS processess the transaction again, determining that a token can be issued

6. The AS issues a token to the RC

7. The RC uses the token with the RS

2. Transaction request

To start a transaction, the RC makes a transaction request to the transaction endpoint of the AS. The RC creates a JSON [RFC8259] document with five primary sections, included as members of a root JSON object.

client Information about the RC making the request, including display name, home page, logo, and other user-facing information. This section is RECOMMENDED.

resources Information about the RS's the resulting token will be applied to, including locations, extents of access, types of data being accessed, and other API information. This section is REQUIRED.

user Information about the RO as known to or provided to the RC, in the form of assertions or references to external data. This section is OPTIONAL.

interact Information about how the RC is able to interact with the RO, including callback URI's and state. This section is REQUIRED if the RC is capable of driving interaction with the user.

keys Information about the keys known to the RC and able to be presented in future parts of the transaction. This section is REQUIRED. (Note: I can't think of a good reason for this to be optional.)

Each section consists of either a JSON object or an array of JSON objects, as described in the subsections below. Many sections MAY be represented by an appropriate handle instead as described in Section 9. In such cases, the section is replaced entirely by the handle presentation, which is a single string instead of a JSON object. The RC MAY present additional sections as defined by extensions of this specification. The AS MUST ignore any sections that it does not understand.

2.1. Client

This section provides descriptive details of the RC software making the call. This section is a JSON object, and all fields are OPTIONAL. The RC MAY send additional fields, and the AS MUST ignore all fields that it does not understand.

name Display name of the RC software

uri User-facing web page of the RC software

logo_uri Display image to represent the RC software

```
client: {

  name: "Display Name",
  uri: "https://example.com/client"

}
```

The AS SHOULD use this information in presenting any authorization screens to the RO during interaction.

The client information MAY instead be presented as a client handle reference Section 9.4.

2.2. Resource

This section identifies what the RC wants to do with the API hosted at the RS. This section is a JSON array of objects, each object representing a single resource or resource set. That AS MUST interpret the request as being for all of the resources listed.

actions The types of actions the RC will take at the RS

locations URIs the RC will call at the RS

data types of data available to the RC at the RS's API

```
resources: [
  {
    actions: ["read", "write"],
    locations: ["https://exapmle.com/resource"]
    data: ["foo", "bar"]

  }
]
```

This can also be presented as a set of resource handle references Section 9.5, or a combination of handles and resource structures.

2.3. User

This section provides a verifiable assertion about the RO interacting with the RC on behalf of the request.

assertion The value of the assertion as a string.

type The type of the assertion. Possible values include "oidc_id_token"...

```
user: {

  assertion: "eyj0....",
  type: "oidc_id_token"

}
```

This can also be presented as a user handle reference Section 9.6.

2.4. Interact

This section provides details of how the RC can interact with the RO. All interact requests MUST have the "type" field.

type REQUIRED. Type of interaction. Can be "redirect" or "device". The "redirect" type indicates that the RO is capable of sending the user to an arbitrary URL to be returned from the AS. The "device" type indicates that the RC is capable of communicating a short, human-readable code to the RO, which the RO can enter interactively to the AS.

callback OPTIONAL. IF the RC is capable of receiving inbound messages from the RO's browser, this indicates the URI to send the RO to after interaction. This URI SHOULD (MUST?) be unique per transaction and MUST be hosted or accessible by the RC. This URI MUST NOT contain any fragment component. This URI MUST be protected by HTTPS, be hosted on a server local to the user's browser ("localhost"), or use an application-specific URI scheme. The allowable URIs MAY be limited by the AS based on the RC's presented key information.

state REQUIRED if the "callback" parameter is used. Unique value to be returned to the application as a query parameter on the callback URL, must be sufficiently random to be unguessable by an attacker. MUST be generated by the RC for this transaction.

Each interaction type has its own parameters and behaviors, detailed below.

This section MUST NOT be represented by a handle reference. (Note: this

2.5. Keys

This section lists the keys that the RC can present proof of ownership. The RC MUST send at least one key. The RC MAY send more than one key, but only one key of each type.

jwks Value of the public key as a JWK Set JSON object [Note: should this be a single JWK instead? And do we want to bother with url-based references?]. 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. This key type MUST be proved using either the detached JWS or HTTP-Signing mechanisms.

cert String serialized value of the certificate thumbprint as per OAuth-MTLS. This key type MUST be proved using Mutual TLS.

did The DID URL identifying the key (or keys) used to sign this request. This key type MUST be proved using [[note -- how?]]

The RC MUST provide proof of possession of all presented keysSection 10. All presented keys MUST be validated by the AS as per the Key Validation section.

This section MAY also be presented as a key handle reference Section 9.7. The keys referenced by a handle MUST be validated by the AS.

The following non-normative example shows all three key methods:

```
key: {

  jwks: { keys: [ alg: RS256, kty: ... ] },
  cert: "MII....",
  did: "did:v:foo...."

}
```

3. Interaction response

When evaluating a transaction request, the AS MAY determine that it needs to have the RO present to interact with the AS before issuing a token. This interaction can include the RO logging in to the AS, authorizing the transaction, providing proof claims, determining if the transaction decision should be remembered for the future, and other items.

The AS responds to the RC based on the type of interaction supported by the RC in the transaction request.

This response can indicate a set of keys are bound to the transaction as in Key Binding. This response includes a transaction handle as in Transaction Handle.

3.1. Redirect interaction

If the RC supports a "redirect" style interaction, the AS creates a unique interaction URL and returns it to the RC. This URL MUST be associated with a single pending transaction.

interaction_url The interaction URL that the RC will direct the RO to. This URL MUST be unique to this transaction request. The URL SHOULD contain a random portion of sufficient entropy so as not to be guessable by the user. The URL MUST NOT contain the transaction handle or any RC identifying information. This URL MUST be protected by HTTPS. This URL MUST NOT contain any fragment component.

handle The transaction handle to use in the continue request once the RO has been returned to the RC via the callback URL. See the section on transaction handlesSection 9.3.

```
{

  interaction_url: "https://server.example.com/interact/123asdfklj",
  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    type: "bearer"
  }
}
```

When the RC receives this response, it MUST launch the system browser, redirect the RO through an HTTP 302 response, display the URL through a scannable barcode, or otherwise send the RO to the interaction URL. The RC MUST NOT modify the interaction URL or append anything to it, including any query parameters, fragments, or special headers.

The interaction URL MUST be reachable from the RO's browser, though note that the RO MAY open the interaction 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.

Upon receiving an incoming request at the interaction URL, the AS MUST determine the transaction associated with this unique URL. If the transaction is not found, an error is returned to the end user through the browser and the AS MUST NOT attempt to redirect to a callback URL. When interacting with the RO, the AS MAY perform any of the behaviors in the User Interaction section Section 5.

3.2. Callback response

If the RC has supplied a callback URL in its interact request Section 2.4, the AS returns the user to the RC by redirecting the RO's browser to the RC's callback URL presented at the start of the transaction, with the addition of two query parameters.

state REQUIRED. The (hashed?) value of the state parameter sent by the RC in the initial interaction request Section 2.4.

interact_handle REQUIRED. A shared secret associated with this interaction. This value MUST be sufficiently random so as not to be guessable by an attacker. This value MUST be associated by the AS with the underlying transaction that is associated to with this interaction.

Upon processing this request to the callback URL, the RC MUST match the state value to the value it sent in the original transaction request. The RC then sends a transaction continuation request with the transaction handle returned in the interaction response and the (hash of?) the interaction handle returned as a query parameter to the callback URL.

The RC sends (the hash of? example here is hashed) the interaction handle as the "interact_handle" field of the transaction continuation requestSection 7, using the transaction handle Section 7 returned in the most recent transaction response from the AS.

```
{
    "handle": "80UPRY5NM33OMUKMKSKU",
    "interact_handle": "CuD9MrpSXVKvvI6dN1awtNLx-
HhZy46hJFDBicG4KoZaCmBofvqPxtm7CDMTsUFuvcmLwi_zUN70cCvalI6ENw"
}
```

3.3. Secondary device interaction

If the RC supports a "device" style interaction, the AS creates a unique interaction code and returns it to the RC. The RC communicates this code to the RO and instructs the RO to enter the code at a URL hosted by the AS.

user_code REQUIRED. A 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.

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

wait RECOMMENDED. The amount of time to wait before polling again, in integer seconds.

handle REQUIRED. The transaction handle to use in the continue request. See the section on transaction handlesSection 9.3.

```
{

  user_code: "ABCD1234"
  user_code_url: "https://server.example.com/device",
  wait: 30,
  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    type: "bearer"
  }
}
```

When the RC receives this response, it MUST communicate the user code to the RO. If possible the RC SHOULD communicate the interaction URL to the user as well.

When the RO enters the unique user code at the user code URL, the AS MUST determine which active transaction is associated with the user code. If a transaction is not found, the AS MUST return an error page to the user and MUST NOT attempt to redirect to a callback URL. The AS MAY use any mechanism to interact with the RO as listed in Section 5.

Note that this method is strictly for allowing the user to enter a code at a static URL. If the AS wishes to communicate a pre-composed URL to the RO containing both the user code and the URL at which to enter it, the AS MUST use the "interaction_url" Section 3.1 redirect mechanism instead as this allows the client to communicate an arbitrary interaction URL to the RO.

4. Wait response

If the AS needs the RC to wait before it can give a definitive response to a transaction continue requestSection 7, the AS replies to the transaction request with a wait response. This tells the RC that it can poll the transaction after a set amount of time.

This response includes a transaction handle as in Transaction Handle Section 9.3.

wait REQUIRED. The amount of time to wait before polling again, in integer seconds.

handle REQUIRED. The transaction handle to use in the continue request. This MUST be a newly-created handle and MUST replace any existing handle for this transaction. See the section on transaction handles.

```
{

  wait: 30,
  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    type: "bearer"
  }

}
```

5. Interaction at the AS

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

- authenticate the RO
- gather identity claims about the RO
- gather consent and authorization from the RO
- allow the RO to modify the parameters of the requested transaction (such as disallowing some requested resources)

When the AS has concluded interacting with the RO, the AS MUST determine if the RC has registered a callback URL and state parameter for this transaction. If so, the AS MUST redirect the RO's browser to the callback URL as described in Section 3. If the AS detects an error condition, such as an unknown transaction, an untrustworthy callback URL, an untrustworthy client, or suspicious RO behavior, the AS MUST return an error to the RO's browser and MUST NOT redirect to the callback URL.

6. Error response

If the AS determines that the token cannot be issued for any reason, it responds to the RC with an error message. This message does not include a transaction handle, and the RC can no longer poll for this transaction. The RC MAY create a new transaction and start again.

error The error code.

```
{
  error: user_denied
}
```

TODO: we should have a more robust error mechanism. Current candidate list of errors:

user_denied The RO denied the transaction request.

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

unknown_transaction The transaction continuation request referenced an unknown transaction.

unknown_handle The request referenced an unknown handle.

7. Transaction continue request

Once a transaction has begun, the AS associates that transaction with a transaction handleSection 9.3 which is returned to the RC in one of the transaction responses Section 3.1, Section 3.3, Section 4. This handle MUST be unique, MUST be associated with a single transaction, and MUST be one time use.

The RC continues the transaction by making a request with the transaction handle in the body of the request. The RC MAY add additional fields to the transaction continuation request, such as the interaction handle return in the callback response Section 3.

handle REQUIRED. The (hash of?) transaction handle indicating which transaction to continue.

interaction_handle OPTIONAL. If the RC has received an interaction handle from the callback response of the interaction URL, the RC MUST include the (hash of?) that handle in its transaction continue request.

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

The RC MUST prove all keys initially sent in the transaction requestSection 2.5 as described in Section 10.

8. Token response

access_token The access token that the RC uses to call the RS. The access token follows the handle structure described in Section 9.

handle The transaction handle to use in the continue requestSection 7 to get a new access token once the one issued is no longer usable. See the section on transaction handlesSection 9.3.

```
key: {

  access_token: {
    value: "08ur4kahfga09u23rnkjasdf",
    type: "bearer"
  },
  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    type: "bearer"
  }

}
```

8.1. Presenting Tokens to the RS

A bearer style access token MUST be presented using the Header method of OAuth 2 Bearer Tokens [RFC6750]. A sha3 style access token is hashed as described in Section 9 and presented using the Header method of OAuth 2 Bearer Tokens [RFC6750].

An access token MAY be bound to any keys presented by the client during the transaction request. A bound access token MUST be presented with proof of the key as described in Section 10.

9. Handle references

A handle in this protocol is a value presented from one party to another as proof that they are the appropriate party for part of the transaction. Handles can be used to reference the transaction as a whole, or one of its constituent parts. When a handle is used to represent a part of a transaction request, the handle presentation replaces the original value. In practical terms, this often means that the values of a transaction request are either an object (when the full value is used) or a single string (when the handle is used).

value The value of the handle as a string.

method The verification method, MUST be one of "bearer" or "sha3".

9.1. Presenting handles

Bearer handles are presented by giving the exact string value of the handle in the appropriate place.

SHA3 handles are validated by taking the SHA3 hash of the handle value and encoding it in Base64URL with no padding, and presenting the encoded value.

9.2. Validating handles

Bearer handles are validated by doing an exact byte comparison of the string representation of the handle value.

SHA3 handles are validated by taking the SHA3 hash of the handle value and encoding it in Base64URL with no padding, and comparing that using an exact byte comparison with the presented value.

9.3. Transaction handles

Transaction handles are issued by the AS to the RC to allow the RC to continue a transaction after every step. A transaction handle MUST be discarded after it is used by both the AS and the RC. A transaction MUST have only a single handle associated with it at any time. If the AS determines that the RC can still continue the transaction after a handle has been used, a new transaction handle will be issued in its place. If the AS does not issue a transaction handle in its response to the RC, the RC MUST NOT continue that transaction.

Transaction handles always represent the current state of the transaction which they reference.

Transactions can be continued by the RC if the AS needs to interact with the ROSection 5 and the RC is expecting a callbackSection 3 or if the AS is still waiting on some external conditionSection 4 while the RC is polling. The transaction MAY also be continued after an access token is issued Section 8 as a means of refreshing an access token with the same rights associated with the transaction.

9.4. Client handles

RC handles stand in for the client section of the initial transaction requestSection 2.1. The AS MAY issue a client handle to a RC as part of a static registration process, analogous to a client ID in OAuth 2, allowing the RC to be associated with an AS-side configuration that does not change at runtime. Such static processes SHOULD be bound to a set of keys known only to the RC software.

Client handles MAY be issued by the RS in response to a transaction request. The AS MAY associate the client handle to the interact, resource, and key handles issued in the same response, requiring them to be used together. When the RC receives this handle, it MAY present the handle in future transaction requests instead of sending its information again.

```
{

  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  },
  client_handle: {
    value: "absc2948afgdkjnasdf9082ur3kjasdfasdf89",
    method: "bearer"
  }

}
```

The RC sends its handle in lieu of the client block of the transaction request:

```
{

  client: "absc2948afgdkjnasdf9082ur3kjasdfasdf89"

}
```

9.5. Resource handles

Resource handles stand in for the detailed resource request in the transaction requestSection 2.2. Resource handles MAY be created by the authorization server as static stand-ins for specific resource requests, analogous to OAuth2 scopes.

Resource handles MAY be issued by the RS in response to a transaction request. When the RC receives this handle, it MAY present the handle

in future transaction requests instead of sending its information again.

```
{

  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  },
  resource_handle: {
    value: "foo",
    method: "bearer"
  }

}
```

The RC sends its handle in lieu of the resource block of the future transaction request:

```
{

  resources: ["foo"]

}
```

Handles and object values MAY be combined.

```
{

  resources: [
    "foo",
    {
      actions: ["read", "write"],
      locations: ["https://exapmle.com/resource"]
      data: ["foo", "bar"]
    }
  ]

}
```

9.5.1. Resource-first

[[Strawman idea:]]

In order to facilitate dynamic API protection, an RS MAY pre-register a resource handle in response to an unauthorized request from the RC. In this scenario, the RS creates a transaction request with no client information but describing the resources being protected [[Note: this

is currently at odds with the required format above, perhaps this should be a special mode or flag? We could still use the "keys" section here though.]] The AS returns a resource handle to the RS, which then communicates both the resource handle and the AS transaction endpoint to the RC. The RC then begins its transaction as normal, using the resource handle as one of perhaps several resources it requests.

9.6. User handles

User handles MAY be issued by the AS in response to validating a specific RO during a transaction and stand in for the user section of a transaction requestSection 2.3. This handle MAY refer to the RO that interacted with the AS, the user presented by claims in the transaction request, or a combination of these. This handle can be used in future transactions to represent the current user, analogous to the persistent claims token of UMA 2.

```
{

  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  },
  user_handle: {
    value: "absc2948afgdkjnasdf9082ur3kjasdfasdf89",
    method: "bearer"
  }

}
```

The RC sends its handle in lieu of the user block of the transaction request:

```
{

  user: "absc2948afgdkjnasdf9082ur3kjasdfasdf89"

}
```

9.7. Key handles

Key handles stand in for the keys section of the initial transaction requestSection 2.5. The AS MAY issue a key handle to a RC as part of a static registration process, allowing the RC to be associated with an AS-side configuration that does not change at runtime.

Key handles MAY be issued by the AS in response to a transaction request. The AS SHOULD bind this handle to the client, resource, and user handles issued in the same response. When the RC receives this handle, it MAY present the handle in future transaction requests instead of sending its information again.

```
{

  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  },
  key_handle: {
    value: "absc2948afgdkjnasdf9082ur3kjasdfasdf89",
    method: "bearer"
  }

}
```

The RC sends its handle in lieu of the client block of the transaction request:

```
{

  key: "absc2948afgdkjnasdf9082ur3kjasdfasdf89"

}
```

When the AS receives a key handle, it MUST validate that the keys referenced by the handle are bound to the current transaction request.

10. Binding Keys

Any keys presented by the RC to the AS or RS MUST be validated as part of the transaction in which they are presented. Any keys bound to the transaction are indicated by the bound_keys section of the transaction response. Any keys referenced in this section MUST be used with all future transaction requests.

10.1. Binding a key to a transaction

All keys presented by the RC in the transaction requestSection 2 MUST be proved in all transaction continuation requestsSection 7 for that transaction. The AS MUST validate all keys presented by the RC or referenced in the transaction.

10.2. Detached JWS

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

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

JWS-Signature: eyj0....

When the AS receives the JWS-Signature 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.

10.3. Validating MTLS

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.

10.4. Validating DID

[[Note: validation of DID-based keys could potentially be either detached JWS or MTLS, depending on the type of key used, or some other validation mechanism.]] The RC signs the request using [some HTTP signing mechanism] and its private key, and attaches the signature to the HTTP request using [a header method?]. [Note: is DID just a key-lookup mechanism here or should we use a different kind of crypto method as well?]

11. Acknowledgements

12. IANA Considerations

This specification creates one registry and registers several values into existing registries.

13. Security Considerations

All requests have to be over TLS or equivalent.

14. Privacy Considerations

Handles are passed between parties and therefore should be stateful and not contain any internal structure or information, which could leak private data.

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)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/bcp195>.

[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>.

[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>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

Appendix A. Document History

- 01

- Made JSON multimodal for handle requests.

- Major updates to normative language and references throughout document.

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

- 00

- Initial submission.

Author's Address

Justin Richer (editor)
Bespoke Engineering

Email: ietf@justin.richer.org