Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 17, 2020

J. Richer, Ed.
Bespoke Engineering

March 16, 2020

Transactional Authorization
draft-richer-transactional-authz-06

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

Copyright Notice

Table of Contents

1. Protocol

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

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 and user code endpoints, and these are introduced to the RC as needed during the transaction process.

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 processes 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.

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.

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.)

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

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

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.

claims Information about the RO as known to or provided to the AS being requested by the RS. This section is OPTIONAL.

capabilities Information about the capabilities of the RC. This section is 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.

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

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

```
           "email": true,
           "oidc_id_token": true
       }
   }
```

2.1. Display

This section provides descriptive details of the RC software making the call, useful for displaying information about the client to the user during the authorization request. 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

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

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

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

2.2. Resource

This section identifies what the RC wants to do with the API hosted at the RS. When requesting a single resource, 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

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

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

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

When requesting multiple resources, this section is a JSON object whose keys identifiers chosen by the RC and whose values are resource description arrays, as described above.

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

When a client requests multiple resources in this manner, a successful response MUST follow the multiple access token format.

2.3. User

This section provides a verifiable assertion about the person interacting with the RC on behalf of the request. This person MAY be the RO or MAY be another party.

assertion The value of the assertion as a string.

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

```
    "user": {
        "assertion":
"eyJraWQiOiIxZTlnZGs3IiwiYWxnIjoiUlMyNTYifQ.ewogImlzcyI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwk
E7yM2macAAgNCUwtik6SjoSUZRcf-
O5lygIyLENx882p6MtmwaL1hd6qn5RZOQ0TLrOYu0532g9Exxcm-
ChymrB4xLykpDj3lUivJt63eEGGN6DH5K6o33TcxkIjNrCD4XB1CKKumZvCedgHHF3IAK4dVEDSUoGlH9z4pP_eWYNXvqQOj
rDaQzUHl6cQQWNiDpWOl_lxXjQEvQ",
        "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 fields are OPTIONAL, and the RC MAY include multiple possible interaction modes. If a field is not present, it is interpreted as negative support for that feature.

callback If this object is present, it indicates the RC is capable of receiving inbound messages from the RO's browser in response to user interaction. This object contains the following fields:

uri REQUIRED. Indicates the URI to send the RO to after interaction. This URI MAY 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. If the RC needs any state information to tie to the front channel interaction response, it MUST encode that into the callback URI. The allowable URIs and URI patterns MAY be limited by the AS based on the RC's presented key information. The callback URI SHOULD be presented to the RO during the interaction phase before redirect.

nonce REQUIRED. Unique value to be used in the calculation of the "hash" query parameter on 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 transaction.

hash_method OPTIONAL. The signature mechanism to be used for the callback hash in Section 3.3. Can be one of sha3 or sha2. If absent, the default value is sha3.

redirect If this is set to true, the RC is capable of redirecting

the RO to an arbitrary interaction URL as described in Section 5.
The RC MAY communicate the URI to the user through a browser
redirection, a QR code, or some other mechanism.

user_code If this is set to true, the RC is capable of displaying a short user code to the user and directing them to a fixed URL as described in Section 5.

didcomm If this is set to true, the RC is capable of relaying a DIDComm message to an agent or wallet.

didcomm_query If this is set to true, the RC is capable of relaying a DIDComm query to an agent or wallet.

This section MUST NOT be represented by a handle reference. (Note: this decision is largely due to the "callback" section being variable per transaction. We could allow a handle but restrict it to non-callback methods -- but in that case, it's simpler and shorter to just send the booleans instead of having a special case.)

The following example is from an RC that can redirect to the interaction endpoint and receive returns on a callback URI:

```
"interact": {
    "redirect": true,
    "callback": {
        "uri": "https://example.com/client/123456",
        "nonce": "VJLO6A4CAYLBXHTR0KRO"
    }
}
```

2.5. Keys

This section lists the keys that the RC can present proof of ownership. The RC MUST send at least one key format. The RC MAY send more than one key format, but all keys MUST be equivalent.

proof The form of proof that the RC will use when presenting the key to the AS. The valid values of this field and the processing requirements for each are detailed in Section 10. This field is REQUIRED.

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.

cert PEM serialized value of the certificate used for TLS transactions, with optional internal whitespace.

cert#256 The certificate thumbprint calculated as per OAuth-MTLS [I-D.ietf-oauth-mtls].

did The DID URL identifying the key (or keys) used to sign this request.

The RC MUST provide proof of possession of all presented keysSection 10. All presented keys MUST be validated by the AS using the method defined by proof.

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 three key types, with the detached JWS proofing mechanism:

```
 "keys": {
     "proof": "jwsd",
     "jwks": {
         "keys": [
             {
                 "kty": "RSA",
                 "e": "AQAB",
                 "kid": "xyz-1",
                 "alg": "RS256",
                 "n": "kOB5rR4Jv0GMeLaY6_It_r3ORwdf8ci_JtffXyaSx8xY..."
             }
         ]
     },
     "cert": "MIIEHDCCAwSgAwIBAgIBATANBgkqhkiG9w0BAQsFA...",
     "did": "did:example:CV3BVVXK2PWWLCRQLRFU#xyz-1"
 }
```

2.6. Capabilities

This section lists the extensions and special features supported by the RC. Values of extensions MUST be listed in [[a registry]].

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

2.7. Claims

This section allows the RC to request identity and authentication information about the RO. All fields are OPTIONAL and consist of boolean values. A "true" value indicates that the RC is requesting that claim be returned in the transaction response. A "false" or

omitted value indicates that the RC is not requesting that claim be returned.

subject A subject identifier for the current user.

email An email address for the current user.

phone A phone number for the current user.

oidc_id_token An OpenID Connect ID Token for the current user.

The following non-normative example shows how to request the subject and email address of the current user.

```
"claims": {
    "subject": true,
    "email": true
}
```

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. The AS MAY respond with multiple possible interaction methods to be chosen by the RC. For example, if the RC indicates that it can handle redirects and user codes and has a callback URI, it would send a transaction request like this:

```
   {
       "interact": {
           "redirect": true,
           "user_code": true,
           "callback": {
               "uri": "https://client.example.net/return/123455",
               "nonce": "LKLTI25DK82FX4T4QFZC"
           }
       },
       "resources": [
           "dolphin-metadata"
       ],
       "key": "7C7C4AZ9KHRS6X63AJAO",
       "display": {
           "name": "My Client Display Name",
           "uri": "https://example.net/client"
       }
   }
```

The AS would then respond with a transaction response like this:

```
{
    "interaction_url": "https://server.example.com/interact/
4CF492MLVMSW9MKMXKHQ",
    "server_nonce": "MBDOFXG4Y5CVJCX821LH",
    "user_code": {
        "url": "https://server.example.com/interact/device",
        "code": "A1BC-3DFF"
    },
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

This response MUST include a transaction handle as described in Section 9.3 so that the transaction can continue after the user has interacted.

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 the current transaction and no other transaction.

interaction_url REQUIRED. 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 REQUIRED. The transaction handle to use in the continue the transaction Section 7.

```
{
    "interaction_url": "https://server.example.com/interact/
4CF492MLVMSW9MKMXKHQ",
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "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. **Interaction URI return**

If the RC has supplied a callback URL in its interact request Section 2.4, the AS returns a nonce in its interaction response.

server_nonce REQUIRED. A unique value from the server included in the calculation of the "hash" value returned in the callback response. REQUIRED if the client has sent a "callback" parameter in its interaction request.

This example also includes the interaction URL from Section 3.1.

```
{
    "interaction_url": "https://server.example.com/interact/
4CF492MLVMSW9MKMXKHQ",
    "server_nonce": "MBDOFXG4Y5CVJCX821LH",
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

When interaction has concluded, 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.

hash REQUIRED. The interaction hash value as described in Section 3.3.

interact_ref 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.

The AS MUST properly process the callback parameter from the interaction request as a URL, adding these values as query parameters. The AS MUST NOT use simple string concatenation. For example, for the callback URL of "https://example.com/client/123456", the AS would add query parameters as follows (newlines added for display purposes only):

```
https://example.com/client/123456
  ?
hash=p28jsq0Y2KK3WS__a42tavNC64ldGTBroywsWxT4md_jZQ1R2HZT8BOWYHcLmObM7XHPAdJzTZMtKBsaraJ64A
  &interact_ref=4IFWWIKYBC2PQ6U56NL1
```

Upon processing this request to the callback URL, the RC MUST calculate the expected value of the "hash" parameter as described in Section 3.3 and compare that value to the "hash" parameter on the incoming request.

The RC also sends (the hash of? example here is not hashed) the interaction reference as the "interact_ref" field of the transaction continuation requestSection 7, using the transaction handle Section 9.3 returned in the most recent transaction response from the AS.

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

3.3. Calculating the interaction hash

The "hash" parameter in the interaction response ties the front channel response to a transaction by using values known only to the parties in the transaction. To calculate the "hash" value for the interaction response, the party doing the calculation first takes the "nonce" value sent by the RC in the interaction section of the initial transaction request Section 2.4, the "server_nonce" value returned in the transaction response Section 3.2, and the "interact_ref" returned in the callback response Section 3.2. 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" section of the interaction request (Section 2.4). If the "hash_method" value is not present in the RC's request, the AS defaults to "sha3".

3.3.1. SHA3

The "sha3" hash method consists of hashing the 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_jZQ1R2HZT8BOWYHcLmObM7XHPAdJzTZMtKBsaraJ64A
```

3.3.2. SHA2

The "sha2" hash method consists of hashing the 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_Bpj84rh4mC9aE9x7HPBFcIHw
```

3.4. Secondary device interaction

If the RC supports a "user_code" style interaction, the AS creates a unique user 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. An object containing the user code information.

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. If not specified, the default is 30 seconds. See Section 4.

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

```
{
    "user_code": {
        "url": "https://server.example.com/interact/device",
        "code": "A1BC-3DFF"
    },
    "wait": 30,
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "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. However, the URL is generally understood to be stable over time for a given service, and this URL MAY be communicated through a static means such as the device's documentation or packaging.

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": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

5. Interaction at the AS

When the RO is interacting with the AS at the interaction uri, 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 nonce 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 3.4, 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 reference return in the callback response Section 3.

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

interaction_ref OPTIONAL. If the RC has received an interaction reference from the callback response of the interaction URL, the RC MUST include the (hash of?) that reference 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.

[[Note: should we allow the client to mutate the transaction at this point? We already allow the presentation of the interaction handle, and any messaging protocols like DIDComm would allow additional work to be done here. But do we want the client to be able to specify additional resources, or new interaction methods, or anything like that? I'm inclined not to so that's been left out for now.]]

8. Transaction 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. This field is mutually exclusive with the multiple_access_tokens field.

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.

claims The set of claims that the AS asserts for the current user. This field is returned only if the RC has requested using the "claims" object. This object contains the following OPTIONAL fields:

updated_at ISO-formatted timestamp string of when the user's profile information was last updated. The RC SHOULD compare

this value to previous values when determining whether to get additional user information from an external endpoint.

subject The machine-readable unique subject identifier of the user.

email The email address of the user.

phone The phone number of the user.

oidc_id_token An OpenID Connect ID Token representing the user.

multiple_access_tokens If the RC requested multiple named resources, this structure contains the access tokens. The keys of this JSON object are the labels determined by the RC in its request, and the values are the same as a single access_token response. This field is mutually exclusive with the access_token field.

display_handle A value used to represent the information in the display object that the client can use in a future request, as described in Section 9.4.

key_handle A value used to represent the information in the key object that the client can use in a future request, as described in Section 9.7.

user_handle A value used to represent the information in the user object that the client can use in a future request, as described in Section 9.6.

display_handle A value used to represent the information in the resource object that the client can use in a future request, as described in Section 9.5.

The following example shows a single access token, a transaction handle, and a set of claims about the current user being returned in a successful response.

```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "type": "bearer"
    },
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    },
    "claims": {
        "subject": "UR64TB8N6BW7OZB8CDFONP-MHKUR6",
        "email": "alice@example.com"
    }
}
```

The following example shows multiple access tokens, a key handle, and a user handle being returned in a successful response. The RC has chosen the labels "token1" and "token2" in this response.

```
{
    "multiple_access_tokens": {
        "token1": {
            "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
            "type": "bearer"
        },
        "token2": {
            "value": "UFGLO2FDAFG7VGZZPJ3IZEMN21EVU71FHCARP4J1",
            "type": "bearer"
        }
    },
    "user_handle": {
        "value": "XUT2MFM1XBIKJKSDU8QM",
        "type": "bearer"
    },
    "key_handle": {
        "value": "7C7C4AZ9KHRS6X63AJAO",
        "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 9.1 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.

8.2. Additional user information

Additional user information MAY be made available to the RC through use of an access token at a protected resource representing the user. This endpoint could be an OpenID Connect UserInfo Endpoint, a SCIM endpoint, or another similar resource. Specification of this resource is outside the scope of this specification.

9. Handles

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.

type 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. Display handles

RC handles stand in for the display section of the initial transaction requestSection 2.1. The AS MAY issue a display 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.

Display handles MAY be issued by the RS in response to a transaction request. The AS MAY associate the display 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": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    },
    "display_handle": {
        "value": "VBUEOIQA82PBY2ZDJW7Q",
        "type": "bearer"
    }
}
```

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

```
{

  "display": "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. In such cases, the resource handle returned represents the total of all resources represented in the request.

```
{
    "wait": 30,
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    },
    "resources_handle": {
        "value": "KLKP36N7GPOKRF3KGH5N",
        "type": "bearer"
    }
}
```

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

```
{

  "resources": ["KLKP36N7GPOKRF3KGH5N"]

}
```

Note that handles and object values MAY be combined in a single request.

```
{
    "resources": [
        {
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        },
        "dolphin-metadata",
        "KLKP36N7GPOKRF3KGH5N"
    ]
}
```

If the RC requests multiple named resources to result in multiple access tokens, the AS MUST NOT return a resource handle.

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.

```
{
    "wait": 30,
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    },
    "user_handle": {
        "value": "XUT2MFM1XBIKJKSDU8QM",
        "type": "bearer"
    }
}
```

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

```
{

  "user": "XUT2MFM1XBIKJKSDU8QM"

}
```

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 display, 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.

```
{
    "wait": 30,
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    },
    "key_handle": {
        "value": "7C7C4AZ9KHRS6X63AJAO",
        "type": "bearer"
    }
}
```

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

```
{

  "keys": "7C7C4AZ9KHRS6X63AJAO"

}
```

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

9.8. Claims handles

Claims handles stand in for the claims section of the initial transaction requestSection 2.7. Claims handles MAY be created by the authorization server as static stand-ins for specific claim set requests, analogous to OIDC scopes.

Claims handles MAY be issued by the RS in response to a transaction request. In such cases, the resource handle returned represents the total of all claims requested in the transaction request. When the RC receives this handle, it MAY present the handle in future transaction requests instead of sending the claims information again.

```
{
    "wait": 30,
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    },
    "claim_handle": {
        "value": "14XF3WKRPKW4RN9AROOC",
        "type": "bearer"
    }
}
```

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

```
{

  "claims": "14XF3WKRPKW4RN9AROOC"

}
```

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

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.The type of binding used is indicated by the proof parameter of the keys section in the transaction request. Values defined by this specification are as follows:

jwsd A detached JWS signature header

mtls Mutual TLS certificate verification

dpop OAuth DPoP key proof header

httpsig HTTP Signing signature header

oauthpop OAuth PoP key proof authentication header

Additional values can be defined by a registry.

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 at each call to the transaction endpoint. The client MUST NOT use a different key during the transaction.

10.1. Detached JWS

This method is indicated by "jwsd" in the "proof" field of a key request. 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 JWS header MUST contain an alg field appropriate for the key identified by kid and MUST NOT be none.

The RC presents the signature in the 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.

10.2. Mutual TLS

This method is indicated by "mtls" in the "proof" field of a key request. 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 [I-D.ietf-oauth-mtls] section 3.

10.3. DPoP

This method is indicated by "dpop" in the "proof" field of a key request. The RC creates a DPoP signature header as described in [I-D.fett-oauth-dpop] section 2.

10.4. HTTP Signing

This method is indicated by "httpsig" in the "proof" field of a key request. The RC creates an HTTP Signature header as described in [I-D.cavage-http-signatures] section 4. The RC MUST calculate and present the Digest header as defined in [RFC3230].

10.5. OAuth PoP

This method is indicated by "oauthpop" in the "proof" field of a key request. 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:

o The at (access token) field MUST be omitted [note: this is in contrast to the requirements in the existing spec]

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

11. Acknowledgements

12. IANA Considerations

[We'll want a registry for key proof types, and maybe some other field names. We'll need to register at least one header and maybe some others?]

13. Security Considerations

All requests have to be over TLS or equivalent. 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

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>.

[I-D.cavage-http-signatures]
Cavage, M. and M. Sporny, "Signing HTTP Messages", draft-cavage-http-signatures-12 (work in progress), October 2019.

[I-D.fett-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)", draft-fett-oauth-dpop-04 (work in progress), March 2020.

[I-D.ietf-oauth-mtls]
Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", draft-ietf-oauth-mtls-17 (work in progress), August 2019.

[I-D.ietf-oauth-signed-http-request]
Richer, J., Bradley, J., and H. Tschofenig, "A Method for Signing HTTP Requests for OAuth", draft-ietf-oauth-signed-http-request-03 (work in progress), August 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

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

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

[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

-06

o Added multiple resource requests and multiple access token response.

-05

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

- Added "capabilities" request for inline discovery support.

- 04

- Added crypto agility for callback return hash.

- Changed "interaction_handle" to "interaction_ref".

- 03

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

- Created signed return parameter for front channel return.

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

- Changed "key" to "keys".

- Separated key proofing from key presentation.

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

- 02

- Minor editorial cleanups.

- 01

- Made JSON multimodal for handle requests.

- Major updates to normative language and references throughout document.

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

- 00

- Initial submission.

Author's Address

Justin Richer (editor)
Bespoke Engineering

Email: ietf@justin.richer.org