Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 30 July 2020

D. Hardt, Ed.
SignIn.Org

27 January 2020

The XAuth Protocol
draft-hardt-xauth-protocol-00

Abstract

Client software often desires resources or identity claims that are managed independent of the client. This protocol allows a user and/or resource owner to delegate resource authorization and/or release of identity claims to an authorization server. Client software can then request access to resources and/or identity claims by calling the authorization server. The authorization server acquires consent and authorization from the user and/or resource owner if required, and then returns the authorization and identity claims that were approved. This protocol can be extended to support alternative client authentication mechanisms, authorizations, claims, and interactions.

[Editor: suggestions on how to improve this are welcome!]

[Editor: suggestions for other names than XAuth are welcome!]

Note to Readers

Source for this draft and an issue tracker can be found at https://github.com/dickhardt/hardt-xauth-protocol (https://github.com/dickhardt/hardt-xauth-protocol).

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 30 July 2020.

Copyright Notice

Table of Contents

1. Introduction

This protocol supports the widely deployed use cases supported by OAuth 2.0 [RFC6749] & [RFC6750], and OpenID Connect [OIDC], an extension of OAuth 2.0, as well as other extensions, and other use cases that are not supported, such as requesting multiple authorizations in one request. This protocol also addresses many of the security issues in OAuth 2.0 by having parameters securely sent

directly between parties, rather than via a browser redirection.

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 asymetric cryptographic functions. Standards have emerged for signing and encrypting tokens with rich payloads (JOSE) that are widely deployed.

Additional use cases are now being served with extensions to OAuth 2.0: OpenID Connect added support for authentication and releasing identity claims; [RFC8252] added support for native apps; [RFC8628] added support for smart devices; and support for [browser_based_apps]

is being worked on. There are numerous efforts on adding proof-of-possession to resource access.

This protocol simplifies the overall architectural model, takes advantage of today's technology landscape, provides support for all the widely deployed use cases, and offers numerous extension points.

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

2. Protocol

2.1. Parties

* *Authorization Server* (AS) - manages Client authorization to API resources and release of identity claims about the User. The AS may require explicit consent from the RO or User to provide these to the Client.

* *Client* - requests authorization to API resources, and/or identity claims about the User. There are two types of Clients: Registered Clients and Dynamic Clients. An AS may support only one or both types. All Clients have a key to authenticate with the AS.

* *Registered Client* - a Client that has registered with the AS and has a Client ID to identify itself, and can prove it possesses a key that is linked to the Client ID. The AS may have different policies for what different Registered Clients can request. The

Client MAY be interacting with a User.

* *Dynamic Client* - a Client that has not been registered with the AS, and each instance will generate it's own key pair so it can prove it is the same instance of the Client on subsequent requests. A single-page application with no server is an example of a Dynamic Client. The Client MUST be interacting with a User.

* *User* - the person who has delegated making identity claims about themselves to the AS, and is interacting with the Client.

* *Resource Server* (RS) - has API resources that require an access token from the AS for access.

* *Resource Owner* (RO) - owns the RS, and has delegated RS access token creation to the AS. The RO may be the same entity as the User, or may be a different entity that the AS interacts with independent of the Client.

2.2. Handles

Handles are strings issued by the AS to the Client, that are opaque to the Client.

* *completion handle* - represents the client request. Presented back to the AS in a completion request.

* *access handle* - represents the access the AS has granted the Client to the RS. The Client proves possession of its key when presenting an access handle to access an RS. The RS is able to understand the authorizations represented by the access handle directly, or through an introspection call. The RS is able to verify the access handle was issued to the Client that presented it.

* *refresh handle* - represents the access the AS has granted the Client to a RS. Presented back to the AS when the Client wants a fresh access token or access handle.

When presenting any of the above handles, the Client always proves possession of its key.

2.3. Reused Terms

* *access token* - an access token as defined in [RFC6749] Section 1.4.

* *Claims* - Claims as defined in [OIDC] Section 5.

* *Client ID* - an AS unique identifier for a Registered Client as defined in [RFC6749] Section 2.2.

* *ID Token* - an ID Token as defined in [OIDC] Section 2.

* *Claims* - Claims as defined in [OIDC] Section 5.

* *NumericDate* - a NumericDate as defined in [RFC7519] Section 2.

2.4. Sequence

1. *AS Discovery* The Client discovers the AS end point, keys, supported claims and authorizations, and other capabilities. Some, or all of this information could be manually preconfigured, or dynamically obtained. Dynamic AS discovery is out of scope of this document.

2. *Initiation Request* (Section 8.7) The Client creates an initiation request (Section 3) for authorization to API resources and/or identity claims about the User and sends it with an HTTP POST to the AS endpoint.

3. *Initiation Response* (Section 8.8) The AS processes the request and determines if it needs to interact with the RO or User. If interaction is not required, the AS returns a completion response (Section 6). If interaction is required the AS returns an initiation response (Section 4) that includes completion handle and uri, and interaction instructions if the AS wants the Client to start the interaction.

4. *Interaction* (Section 5) If the AS sent interaction instructions to the Client, the Client executes them. The AS then interacts

with the User and/or RO to obtain authorization. The AS interaction with an RO may be asynchronous, and take time to complete.

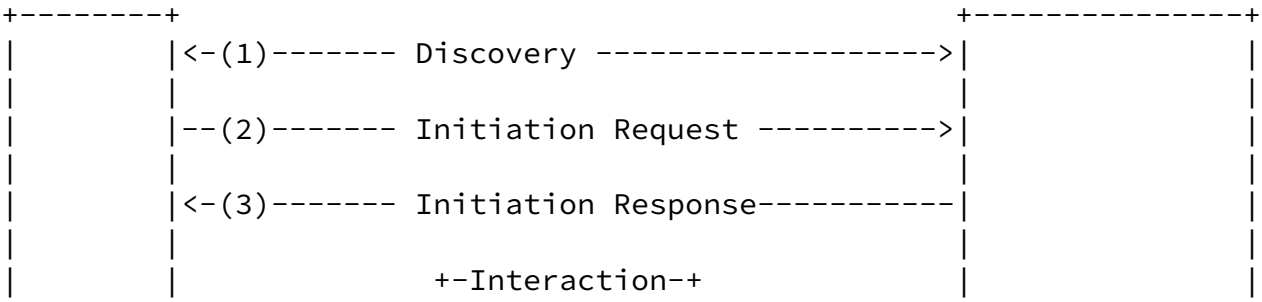
5. *Completion Request* (Section 8.10) The Client sends the completion handle to the completion uri. The Client may make the completion request while waiting for the interaction to complete.

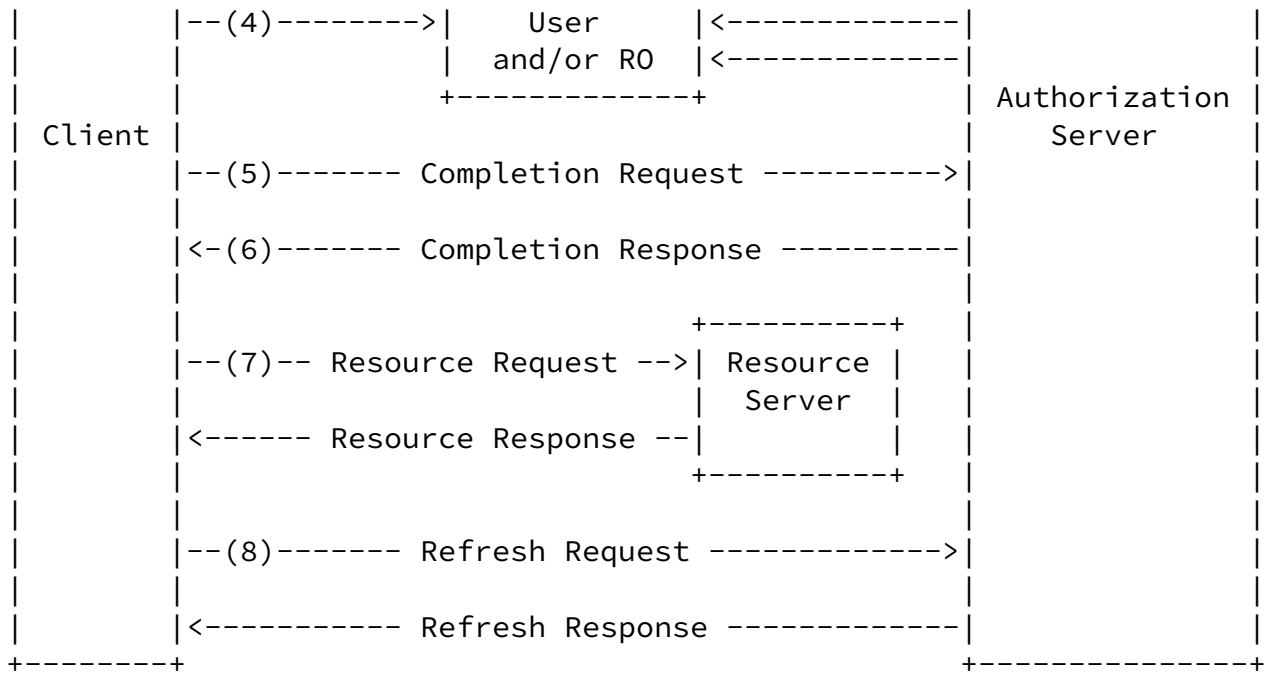
6. *Completion Response* (Section 8.11) When any required interaction has been completed, the AS responds to the Client with a completion response (Section 6) containing authorized RS access tokens and User identity claims. The AS may provide refresh handles and uris for each access token if they are authorized. If proof-of-possession is required for accessing the RS, the AS will provide access handles instead of access tokens. If the AS has not completed the interaction, it will return a retry response for the Client to make a new completion request.

7. *Resource Request* (Section 8.12 & Section 8.13) The Client uses an access token with the RS, or the access handle if proof-of-possession is required for access.

8. *Access Refresh* (Section 8.14) If the Client received a refresh handle and uri, it sends the refresh handle to the refresh uri, and receives a fresh access token or handle.

Sequence Diagram





3. Initiation Request JSON

Following is a non-normative JSON [RFC8259] document example where the Client wants to interact with the User with a popup and is requesting identity claims about the User and read access to the User's contacts:

Example 1

```
{
    "as"    :"https://as.example",
    "iat"   :"1579046092",
    "nonce" :"f6a60810-3d07-41ac-81e7-b958c0dd21e4",
    "client": {
        "display": {
            "name"  :"SPA Display Name",
            "uri"   :"https://spa.example/about"
        },
        "interaction": {
            "type"  :"popup"
        }
    },
    "authorizations": {
        "type"      :"oauth_scope",
        "scope"     :"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 where the Client has previously authenticated the User, and is requesting additional authorization:

Example 2

```
{
   "as"     :"https://as.example",
    "iat"   :"1579046092",
    "nonce" :"0d1998d8-fbfa-4879-b942-85a88bff1f3b",
    "client": {
        "id"          : "di3872h34dkJW",
        "interaction": {
            "type"  : "redirect",
            "uri"   : "https://web.example/return"
        }
    },
    "user": {
        "identifiers": {
            "oidc": {
                "iss"   :"https://as.example",
                "sub"   :"123456789"
            }
        }
    },
    "authorizations": {
        "type"  :"oauth_scope",
        "scope" :"read_calendar write_calendar"
    }
}
```

3.1. Top Level Attributes

as - the unique string identifier for the AS. This attribute is REQUIRED.

iat - the time of the request as a NumericDate.

nonce - a unique identifier for this request. This attribute is REQUIRED. Note the completion response MUST contain a matching nonce attribute.

3.2. "client" Object

The client object MUST contain either the client_id attribute for Registered Clients, or the display object for Dynamic Clients. If the Client can interact with the User, then an interaction object MUST be included.

client_id - the identifier the AS has for the Registered Client.

display - the display object contains the following attributes:

* *name* - a string that represents the Dynamic Client

[Editor: a max length for the name?]

* *uri* - a URI representing the Dynamic Client

[Editor: a max length for the name?]

The name and uri will be displayed by the AS when prompting for authorization.

interaction - the interaction object contains the type of interaction the Client will provide the User. Other attributes are dependent on the interaction type value.

* *type* - contains one of the following values. Types are listed from highest to lowest fidelity. The interaction URI is the value returned by the AS in the initiation response interaction object Section 4.2, if a User interaction is required by the AS.

  - *popup* - the Client will load the interaction URI in a modal popup window. The AS will close the window when the interaction is complete.

  - *redirect* - the Client will redirect the user agent to the interaction URI provided by the AS. The AS will redirect to the redirect_uri when the interaction is completed.

  - *qrcode* - the Client will convert the interaction URI to a QR Code per [QR Code] and display it to the User, along with a text message. The User will scan the QR Code and/or follow the message instructions.

* *redirect_uri* - this attribute is included if the type is "redirect". It is the URI that the Client requests the AS to redirect the User to after the AS has completed interacting with the User. If the Client manages session state in URIs, then the redirect_uri should contain that state.

* *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.

[Editor: do we need max pixels or max chars for qrcode interaction? Either passed to AS, or max specified values here?]

[Editor: other possible interaction models could be a "webview",

where the Client can display a web page, or just a "message", where the client can only display a text message]

[Editor: we may need to include interaction types for iOS and Android as the mobile OS APIs evolve.]

[Editor: does the Client include parameters if it wants the completion response signed and/or encrypted?]

3.3. "user" Object

The user object is optional.

identifiers - the identifiers 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 per [OIDC] Section 2.

The user and identifiers objects MAY be included to improve the user experience by the AS. The AS MUST authenticate the User independent of these values.

discovery - MUST contain the JSON true value. Indicates the Client requests the AS to return a "discovered" attribute in the initiation response if the AS has a User at the AS with one or more of the identifiers. This attribute is OPTIONAL. Supporting by the AS of

the discovery attribute is OPTIONAL. The AS MAY return the TBD error if discovery is not supported, or ignore the request.

3.4. "authorizations" Object

The optional authorizations object contains a dictionary of resource objects the Client is requesting authorization to access. The authorizations object may contain one or more of:

* *type* - one of the following values: "oauth_scope", "oauth_rich", or "oauth_rich_list". See Section 3.6 for details.

* *scope* - a string containing the OAuth 2.0 scope per [RFC6749] section 3.3. Present if type is "oauth_scope" or "oauth_rich".

* *authorization_details* - an authorization_details object per [RAR]. Present if type is "oauth_rich".

* *list* - an array of objects containing "scope" and "authorization_details". Present if type is "oauth_rich_list". Used when requesting multiple access tokens and/or handles.

[Editor: details may change as the [RAR] document evolves]

3.5. "claims" Object

The optional claims object contains one or more identity claims being requested. The claims may contain:

* *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.

* vc - [Editor: define how W3C Verifiable Credentials [W3C_VC] can be requested.]

3.6. Authorization Types

* *oauth_scope* - an OAuth 2.0 authorization request per [RFC6749] section 3.3

* *oauth_rich* - a rich authorization request per [RAR]

* *oauth_rich_list* - a list of rich authorization requests

4. Initiation Response JSON

A non-normative example of an initiation response follows:


```
{
    "user": {
        "discovered": true
    },
    "interaction": {
        "type"   : "popup",
        "uri"    : "https://as.example/endpoint/ey5gs32..."
    },
    "completion": {
        "handle" : "eyJhb958.example.completion.handle.9yf3szM",
        "uri"    : "https://as.example/completion/ey7snHGs",
        "wait"   : "10"
    }
}
```

4.1. "user" Object

If the initiation request included a discovery attribute, then the AS

MAY return a "user" object with the "discovered" property set to the JSON value true if one or more of the identifiers provided in the initiation request identify a User at the AS, or the JSON value false if not.

[Editor: reference a security consideration for this functionality.]

4.2. "interaction" Object

If the AS wants the Client to start the interaction, the AS MUST select one of the interaction mechanisms provided by the Client in the initiation request, and include the matching attribute in the interaction object:

* *type* - this MUST match the type provided by the Client in the initiation request client.interaction object.

* *uri* - the URI to interact with the User per the type. This may be a temporary short URL if the type is qrcode so that it is easy to scan.

* *message* - a text string to display to the User if type is qrcode.

[Editor: do we specify a maximum length for the uri and message so that a device knows the maximum it needs to support? A smart device may have limited screen real estate.]

4.3. "completion" Object

The completion object has the following attributes:

* *handle* - the completion handle. This attribute is REQUIRED.

* *uri* - the completion URI. This attribute is REQUIRED.

* *wait* - the amount of time in integer seconds the Client MUST wait before making the completion request. This attribute is OPTIONAL.

5. Interaction Types

If the AS wants the Client to initiate the interaction with the User, then the AS will return an interaction object Section 4.2 so that the Client can can hand off interactions with the User to the AS. The Client will initiate the interaction with the User in one of the following ways:

5.1. "popup" Type

The Client will create a new popup child browser window containing the value of the uri attribute of the interaction object. [Editor: more details on how to do this]

The AS will close the window when the interactions with the User are complete. [Editor: confirm AS can do this still on all browsers, or does Client need to close]

The AS MAY respond to an outstanding completion request Section 8.10 before the popup window has been closed.

5.2. "redirect" Type

The Client will redirect the User to the value of the uri attribute of the interaction object. When the AS interactions with the User are complete, the AS will redirect the User to the redirect_uri the Client provided in the initiation request.

If the Client made a completion request when starting the interaction, the AS MAY respond to the completion request Section 8.10 before the User has been redirected back to the Client.

5.3. "qrcode" Type

The Client will create a [QR Code] of the uri attribute of the interaction object and display the resulting graphic and the message

attribute of the interaction object as a text string.

6. Completion Response JSON

Example non-normative completion response JSON document for Example 1 in Section 3:

```
{
    "iat":"15790460234",
    "nonce":"f6a60810-3d07-41ac-81e7-b958c0dd21e4",
    "authorizations": {
        "type"          : "oauth_scope",
        "scope"         : "read_contacts",
        "expires_in"    : "3600",
        "method"        : "bearer",
        "token"         : "eyJJ2D6.example.access.token.mZf9p"
    },
    "claims": {
        "oidc": {
            "id_token"      : "eyJhbUzI1N.example.id.token.YRw5DFdbW",
            "userinfo" : {
                "name"      : "John Doe",
                "picture"   : "https://photos.example/p/eyJzdkiO"
            }
        }
    }
}
```

Example non-normative completion response JSON document for Example 2 in Section 3:

```
{
    "iat"   :"15790460234",
    "nonce" :"0d1998d8-fbfa-4879-b942-85a88bff1f3b",
    "authorizations": {
        "type"          : "oauth_scope",
        "scope"         : "read_calendar write_calendar",
        "expires_in"    : "3600",
        "method"        : "pop",
        "access": {
            "handle"    : "ey.example.access.handle.9yf3szM",
            "jwk": {
                "x5u"   : "https://as.example/jwk/VBUEOIQA82"
            }
        },
        "refresh": {
            "handle"    : "ey.example.refresh.handle.Jl4FzM",
            "uri"       : "https://as.example/refresh/eyj34"
        }
    }
}
```

Details of the JSON document:

6.1. Top Level Attributes

iat - the time of the response as a NumericDate.

nonce - the nonce that was included in the initiation request JSON Section 3.

6.2. "authorizations" Object

There is an authorizations object in the completion response if there was an authorizations object in the initiation request.

* *type* - the type of claim request: "oauth_scope", "oauth_rich", or "oauth_rich_list". See Section 3.6 for details.

* *list* - an array of objects if the type was "oauth_rich_list". The objects have the following attributes just as if the type was "oauth_rich"

* *scope* - the scopes the Client was granted authorization for. This will be all, or a subset, of what was requested.

* *authorization_details* - the authorization details granted. Only returned for "oauth_rich" and "oauth_rich_list".

* *method* - the access method: "bearer" or "pop". See Section 6.4 for details.

* *token* - an access token for accessing the resource(s). Included if the access method is "bearer".

* *expires_in* - an optional value specifying how many seconds until the access token or handle expire.

* *refresh* - an optional object containing parameters required to refresh an access token or handle. See refresh request Section 8.14.

  - *handle* - an refresh handle used to create the JSON refresh token.

  - *uri* - the refresh uri the Client will use to refresh.

* *access* - an object containing the parameters needed to access resources requiring proof-of-possession. Included if the access method is "pop".

  - *handle* - the access handle.

  - *jwk* - the jwk object to use.

6.3. "claims" Object

There is a claims object in the completion response if there was a claims object in the initiation request.

* *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.

* *vc*

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

6.4. Access Types

There are two types of access to an RS:

* *bearer* - the AS provides a bearer access token that the Client can use to access an RS per Section 8.12.

* *pop* - the AS provides an access handle that the Client presents in a proof-of-possession RS access request per Section 8.13.

7. Discovery

The Client obtains the following metadata about the AS prior to initiating a request:

Endpoint - the endpoint of the AS.

"as" - the unique string identifier for the AS.

Algorithms - the asymetric cryptographic algorithms supported by the AS.

Authorizations - the authorizations the Client may request, if any.

Identity Claims - the identity claims the Client may request, if any, and what public keys the claims will be signed with.

The client may also obtain information about how the AS will sign and/or encrypt the completion response, as well as any other metadata required for extensions to this protocol, as defined in those extension specifications.

8. JOSE Client Authentication

The default mechanism for the Client to authenticate to the AS and the RS is signing a JSON document with JWS per [RFC7515]. The resulting tokens always use compact serialization.

It is expected that extensions to this protocol that specify a different mechanism for the Client to authenticate, would over ride this section.

The completion request JSON is signed with JWS and passed as the body of the POST.

The completion, refresh, and access handles are signed with JWS resulting in JOSE completion, refresh, and access tokens

respectively. These JOSE tokens are passed in the HTTP Authorization header with the "JOSE" parameter per Section 8.5.

The Client will use the same private key to create all tokens.

The Client and the AS MUST both use HTTP/2 ([RFC7540]) or later, and TLS 1.3 ([RFC8446]) or later, when communicating with each other.

[Editor: too aggressive to mandate HTTP/2 and TLS 1.3?]

8.1. JOSE Headers

A non-normative example of a JOSE header for a Registered Client using a key id to identify the Client's public key:

```
{
    "alg":"ES256",
    "typ":"JOSE",
    "kid":"1"
}
```

A non-normative example of a JOSE header for a Registered Client using a certificate to assert the Client's public key:

```
{
    "alg":"ES256",
    "typ":"JOSE",
    "jwk":
        {"kty":"RSA",
        "use":"sig",
        "kid":"1b94c",
        "n":"vrjOfz9Ccdgx5nQudyhdoR17V-IubWMeOZCwX_jj0hgAsz2J_pqYW08
        PLbK_PdiVGKPrqzmDIsLI7sA25VEnHU1uCLNwBuUiCO11_-7dYbsr4iJmG0Q
        u2j8DsVyT1azpJC_NG84Ty5KKthuCaPod7iI7w0LK9orSMhBEwwZDCxTWq4a
        YWAchc8t-emd9qOvWtVMDC2BXksRngh6X5bUYLy6AyHKvj-nUy1wgzjYQDwH
        MTplCoLtU-o-8SNnZ1tmRoGE9uJkBLdh5gFENabWnU5m1ZqZPdwS-qo-meMv
        VfJb6jJVWRpl2SUtCnYG2C32qvbWbjZ_jBPD5eunqsIo1vQ",
        "e":"AQAB",
        "x5c":
        ["MIIDQjCCAiqgAwIBAgIGATz/FuLiMA0GCSqGSIb3DQEBBQUAMGIxCzAJB
        gNVBAYTAlVTMQswCQYDVQQIEwJDTzEPMA0GA1UEBxMGRGVudmVyMRwwGgYD
        VQQKExNQaW5nIElkZW50aXR5IENvcnAuMRcwFQYDVQQDEw5CcmlhbiBDYW1
        wYmVsbDAeFw0xMzAyMjEyMzI5MTVaFw0xODA4MTQyMjI5MTVaMGIxCzAJBg
```

```
            NVBAYTAlVTMQswCQYDVQQIEwJDTzEPMA0GA1UEBxMGRGVudmVyMRwwGgYDV
            QQKExNQaW5nIElkZW50aXR5IENvcnAuMRcwFQYDVQQDEw5CcmlhbiBDYW1w
            YmVsbDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAL64zn8/QnH
            YMeZ0LncoXaEde1fiLm1jHjmQsF/449IYALM9if6amFtPDy2yvz3YlRij66
            s5gyLCyO7ANuVRJx1NbgizcAblIgjtdf/u3WG7K+IiZhtELto/A7Fck9Ws6
            SQvzRvOE8uSirYbgmj6He4iO8NCyvaK0jIQRMMGQwsU1quGmFgHIXPLfnpn
            fajr1rVTAwtgV5LEZ4Iel+W1GC8ugMhyr4/p1MtcIM42EA8BzE6ZQqC7VPq
            PvEjZ2dbZkaBhPbiZAS3YeYBRDWm1p1OZtWamT3cEvqqPpnjL1XyW+oyVVk
            aZdklLQp2Btgt9qr21m42f4wTw+Xrp6rCKNb0CAwEAATANBgkqhkiG9w0BA
            QUFAAOCAQEAh8zGlfSlcI0o3rYDPBB07aXNswb4ECNIKG0CETTUxmXl9KUL
            +9gGlqCz5iWLOgWsnrcKcY0vXPG9J1r9AqBNTqNgHq2G03X09266X5CpOe1
            zFo+Owb1zxtp3PehFdfQJ610CDLEaS9V9Rqp17hCyybEpOGVwe8fnk+fbEL
            2Bo3UPGrpsHzUoaGpDftmWssZkhpBJKVMJyf/RuP2SmmaIzmnw9JiSlYhzo
            4tpzd5rFXhjRbg4zW9C+2qok+2+qDM1iJ684gPHMIY8aLWrdgQTxkumGmTq
            gawR+N5MDtdPTEQ0XfIBc2cJEUyMTY5MPvACWpkA6SdS4xSvdXK3IVfOWA==
            }
}
```

[Editor: the jwk above was copy and pasted from the JWK example. Replace?]

The certificate could be signed by the AS, allowing the AS to verify the signature using the AS public key, or the certificate could be signed by a private key the AS has bound to the Registered Client, allowing each instance of the Registered Client to have its own asymetric key pair.

A non-normative example of a JOSE header for a Dynamic Client

including the public key generated by the Client that matches its its private key:

```
{
    "alg":"ES256",
    "typ":"JOSE",
    "jwk":{
        "kty":"EC",
        "crv":"P-256",
        "x":"Kgl5DJSgLyV-G32osmLhFKxJ97FoMW0dZVEqDG-Cwo4",
        "y":"GsL4mOM4x2e6iON8BHvRDQ6AgXAPnw0m0SfdlREV7i4"
    }
```

```
}
```

A non-normative example of a JOSE Access Token JOSE header for a Client accessing an RS that requires proof-of-possession:

```
{
    "alg":"ES256",
    "typ":"JOSE",
    "jwk":{
        "x5u":"https://as.example/jwk/VBUEOIQA82"
    }
}
```

The "jwk" object in a JOSE access token Section 8.4 MUST be the AS jwk object the AS provided with the access handle.

This decouples how the AS communicates the Client's public key to the RS from how the AS asserts the Client's public key. The RS can have a consistent mechanism assert the Client's public key across all Clients.

One advantage of this is the AS can create a certificate of a Dynamic Client's public key, and pass it by value or reference to the Client to present to the RS.

All JOSE headers MUST have: + the "alg" attribute. + the "typ" attribute set to "jose". + either a "kid" or "jwk" attribute.

[Editor: should we use indicate the type of token (completion, refresh, access) using "typ" or "cty"?]

8.2. JOSE Completion Token

A non-normative example of a payload follows:


```
{
    "as"    :"https://as.example",
    "type"  :"completion",
    "iat"   :"1579046092",
    "jti"   :"f6d72254-4f23-417f-b55e-14ad323b1dc1",
```

```
    "handle":"eyJhb958.example.completion.handle.9yf3szM"
}
```

The payload of the completion token contains:

as - the unique string identifier for the AS.

type - the string "completion", indicating the type of token.

iat - the time the completion token was created as a NumericDate.

jti - a unique identifier for the completion token per [RFC7519] section 4.1.7.

handle the completion handle the AS provided the Client in the initiation response Section 4.

8.3. JOSE Refresh Token

A non-normative example of a payload follows:

```
{
    "as"    :"https://as.example",
    "type"  :"refresh",
    "iat"   :"1579049876",
    "jti"   :"4342046d-83c4-4725-8c72-e9a49245f791",
    "handle":"eyJhb958.example.refresh.handle.9yf3szM"
}
```

The payload of the completion token contains:

as - the unique string identifier for the AS.

type - the string "refresh", indicating the type of token.

iat - the time the JOSE refresh token was created as a NumericDate.

jti - a unique identifier for the JOSE refresh token.

handle the refresh handle the AS provided the Client in the completion response Section 6 or access refresh response Section 8.14.

8.4. JOSE Access Token

The "jwk" object in a JOSE access token header MUST be set to the "jwk" value the AS provided for the access handle.

A non-normative example of a payload follows:

```
{
    "type"  :"access",
    "iat"   :"1579046092",
    "jti"   :"5ef47057-08f9-4763-be8d-162824d43dfb",
    "handle":"eyJhb958.example.access.handle.9yf3szM"
}
```

The payload of the JOSE access token contains:

type - the string "access", indicating the type of token.

iat - the time the JOSE access token was created as a NumericDate.

jti - a unique identifier for the JOSE access token.

handle the access handle the AS provided the Client in the completion response Section 6 or access refresh response Section 8.14.

8.5. HTTP Authorization JOSE Header

The Client authenticates requests by setting the HTTP Authorization header to include the "JOSE" parameter, followed by one or more space characters, followed by the appropriate JOSE token.

A non-normative example:

```
Authorization: JOSE eyJhb.example.completion.token.haDwskpFDBW
```

The JOSE completion token, JOSE refresh token, and the JOSE access token are all passed in this manner.

8.6. JOSE Token Verification

TBD

8.7. Initiation Request

The Client creates a JSON document per Section 3, signs it using JWS [RFC7515], and sends the JWS token to the AS end point using HTTP POST, with a content-type of application/jose.

* *Payload Encryption*

The AS may require the initiation request to be encrypted. If so, the JWS token is encrypted per JWE [RFC7516] using the public key and algorithm specified by the AS.

8.8. Initiation Response

If no interaction is required the AS will return a completion response per Section 8.11, otherwise the AS will return an initiation response per Section 4.

If the AS wants the Client to start the interaction, an interaction object will be returned in the response.

* *Error Responses*

The AS MAY respond with one of the following errors defined in Section 9:

TBD

8.9. Deletion Request

The Client MAY delete an outstanding request using the completion token by making an HTTP DELETE call to the completion uri, setting the HTTP Authorization header per Section 8.5 with the JOSE completion token.

A non-normative deletion request example:

```
DELETE /completion/ey7snHGs HTTP/2
Host: as.example
Authorization: JOSE eyJhb.example.completion.token.haDwskpFDBW
```

* *Error Responses*

The AS MAY respond with one of the following errors defined in Section 9:

TBD

8.10. Completion Request

The Client makes an HTTP GET call to the completion uri, setting the HTTP Authorization header per Section 8.5 with the JOSE completion token.

A non-normative completion request example:

```
GET /completion/ey7snHGs HTTP/2
Host: as.example
Authorization: JOSE eyJhb.example.completion.token.haDwskpFDBW
```

8.11. Completion Response

The AS verifies the JOSE completion token, and then provides a response according to what the User and/or RO have authorized if required. If no signature or encryption was required, the AS will respond with a JSON document with content-type set to application/json.

* *Response Signing*

The AS MAY sign the response with a JWS per [RFC7515] and the private key matching the public key the AS defined as its completion response signing key. The AS will respond with the content-type set to application/jose.

* *Response Encryption*

The AS MAY encrypt the response using the public key provided by the Client, using JWE per [RFC7516]. The AS will respond with the content-type set to application/jose.

* *Error Responses*

The AS MAY respond with one of the following errors defined in Section 9:

TBD

8.12. Bearer Token RS Access

If the access method in the completion response authorizations object Section 6.2 was "bearer", then the Client accesses the RS per Section 2.1 of [RFC6750]

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

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

* *Error Responses*

TBD

8.13. Proof-of-possession RS Access

If the access method in the completion response authorizations object Section 6.2 was "pop", then the Client creates a JOSE access token per Section 8.4 for each call to the RS, setting the HTTP Authorization header per Section 8.5 with the JOSE access token.

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

```
GET /calendar HTTP/2
Host: calendar.example
Authorization: JOSE eyJhbG.example.JOSE.access.token.kwwQb958
```

* *Error Responses*

TBD

8.14. Access Refresh

If the Client received a refresh handle and uri from the AS in the initiation response, and it wants a fresh access token or handle, it creates a JOSE refresh token per Section 8.3. setting the HTTP Authorization header per Section 8.5 with the JOSE refresh token. The AS will then respond with a refresh response.

* *Refresh Response*

A non-normative example refresh response with an access handle:

```
{
    "type"          : "oauth_scope",
    "scope"         : "read_calendar write_calendar",
    "expires_in"    : "3600",
    "method"        : "pop",
    "access": {
        "handle"    : "ey.example.access.handle.9yf3iWszM",
        "jwk": {
            "x5u"   : "https://as.example/jwk/VBUEOIQA82"
        }
    },
    "refresh": {
        "handle"    : "ey.example.refresh.handle.4SkjIi",
        "uri"       : "https://as.example/refresh/eyJl4FzM"
    }
}
```


The refresh response is the same as the authorizations object Section 6.2 in the completion response.

If a new refresh handle and/or refresh uri is returned, the Client MUST use the new refresh handle and/or refresh uri

[Editor: are there other results relevant for [RAR]?]

* *Error Responses*

The AS MAY respond with one of the following errors defined in Section 9:

TBD

9. Error Messages

\[Editor: return "wait" time in completion response when AS wants Client to
The Client MUST generate a fresh completion token when retrying the completi
]

TBD

10. AS Initiated Sequence

[Editor: AS initiated flows have been challenging to implement as OAuth 2.0 did not directly support them, so deployments bounced back and forth between the Client and AS to create a Client initiated flow. Here is a proposal to support AS initiated: authentication; just-in-time (JIT) provisioning; and authorization]

In this sequence the User starts at the AS, and then based on User input, the AS redirects the User to a Client, passing an initiation token Section 10.1, and then the Client retrieves authorizations and/or identity claims about the User. The Client MUST be a Registered Client. The sequence follows:

1. The User is interacting at the AS and wants to use the Client, and provide the Client identity claims and/or authorizations from the AS that the Client has pre-configured.

2. The AS creates a completion handle and uri representing the identity claims and authorizations to be provided to the Client. The AS creates an initiation token containing the AS identifier, the completion handle, and the completion uri.

3. The AS redirects the User to a URI the Client has configured to

accept initiation tokens, passing the initiation token as a query parameters with the name "token".

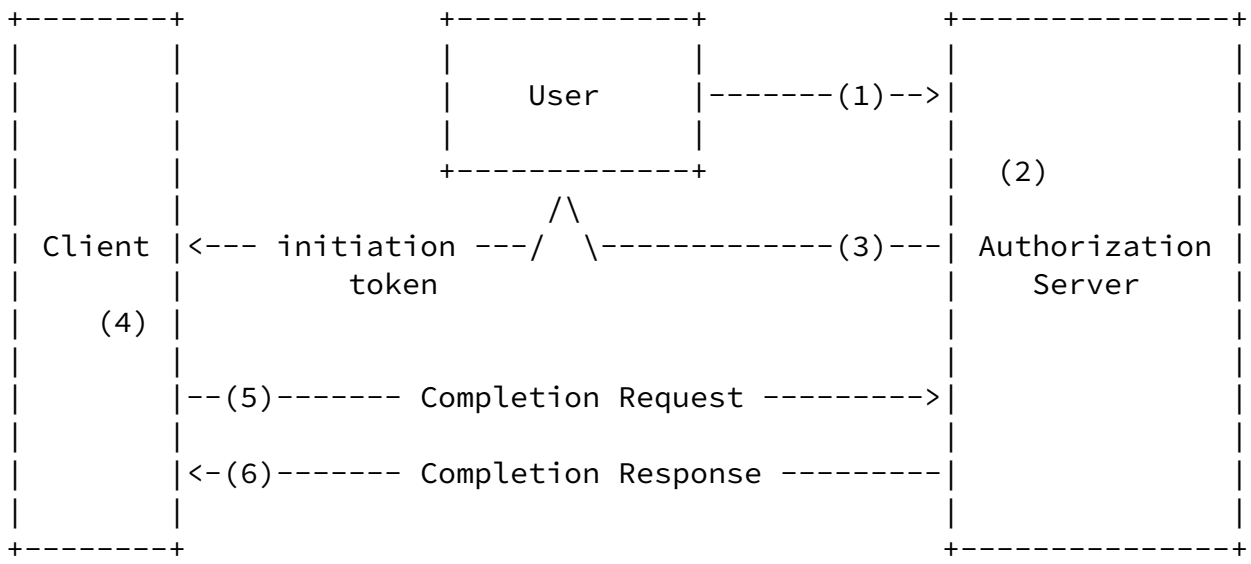
4. The Client verifies the initiation token.

5. The Client makes a completion request per Section 8.10.

6. The AS responds with a completion response JSON document Section 6 per Section 8.11.

The Client now has the User identity claims and/or authorizations. If Client policy permits, the Client can perform JIT provisioning if the User is new to the Client.

AS Initiated Sequence Diagram



10.1. Initiation Token

A non-normative example of an initiation token payload follows:

```
{
    "as": "https://as.example",
    "completion": {
        "handle" : "eyJhb958.example.completion.handle.9yf3szM",
        "uri"    : "https://as.example/completion/ey7snHGs"
    }
}
```

* *as* - the "as" identifier for the AS. This attribute is REQUIRED.

* *completion* - the completion object has the following attributes:

  - *handle* - the completion handle. This attribute is REQUIRED.

  - *uri* - the completion URI. This attribute is REQUIRED.

The initiation token is signed with JWS and uses the compact serialization.

11. Extensibility

This standard can be extended in a number of areas:

* *Client Authentication Mechanisms*

An extension could define other mechanisms for the Client to authenticate and replace JOSE in Section 8 with 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.

* *Initiation Request*

An additional top level object could be added to the initiation request payload if the AS can manage delegations other than authorizations or claims, or some other functionality.

* *"client" Object*

Additional information about the Client that the AS would require related to an extension.

* *"user" Object*

Additional information about the Client that the AS would require related to an extension.

* *"authorizations" Object*

Additional types of authorizations in addition to OAuth 2.0 scopes and RAR.

* *"claims" Object*

Additional types of identity claims in addition to OpenID Connect claims and Verified Credentials.

* *Interaction*

Additional mechanisms for the Client to start an interaction with the User.

* *Access Methods*

Additional mechanisms for the Client to present authorization to a resource.

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

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

12. Rational

1. *Why is there only one mechanism for the Client to authenticate with the AS? Why not support other mechanisms?*

   Having choices requires implementers to understand which choice is preferable for them. Having one default mechanism in the document for the Client to authenticate simplifies most implementations. Extensions can specify other mechanisms that are preferable in specific environments.

2. *Why is the default Client authentication JOSE rather than MTLS?*

   MTLS cannot be used today by a Dynamic Client. MTLS requires the application to have access below what is typically the application layer, that is often not available on some platforms. JOSE is done at the application layer. Many AS deployments will be an application behind a proxy performing TLS, and there are risks in the proxy passing on the results of MTLS.

3. *Why is the default Client authentication JOSE rather than HTTP signing?*

   There is currently no widely deployed open standard for HTTP signing. Additionally, HTTP signing requires passing all the relevant parts of the HTTP request to downstream services within an AS that may need to independently verify the Client identity.

4. *What are the advantages of using JOSE for the Client to authenticate to the AS and a resource?*

Both Registered Clients and Dynamic Clients can have a private key, eliminating the public Client issues in OAuth 2.0, as a Dynamic Client can create an ephemeral key pair. Using asymetric cryptography also allows each instance of a Registered Client to have its own private key if it can obtain a certificate binding its public key to the public key the AS has for the Client. Signed tokens can be passed to downstream components in a AS or RS to enable independent verification of the Client and its request. The AS Initiated Sequence Section 10 requires a URL safe parameter, and JOSE is URL safe.

5. *Why does the AS not return parameters to the Client in the redirect url?*

   Passing parameters via a browser redirection is the source of many of the security risks in OAuth 2.0. It also presents a challenge for smart devices. In this protocol, the redirection from the Client to the AS is to enable the AS to interact with the User, and the redirection back to the Client is to hand back interaction control to the Client if the redirection was a full browser redirect. Unlike OAuth 2.0, the identity of the Client is independent of the URI the AS redirects to.

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

   In OpenID Connect, obtaining claims over the redirect or the Token Endpoint are problematic. The redirect is limited in size, and is not secure. The Token Endpoint is intended to return tokens, and is limited by the "application/x-www-form-urlencoded" format. A UserInfo endpoint returns "application/json", and can return rich claims, just as the completion uri in this protocol.

   [Editor: is there some other reason to have the UserInfo endpoint? What are industry best practices now?]

7. *Why is there still a Client ID? Could we not use a fingerprint of the public key to identify the Client?*

   Some AS deployments do not allow calls from Registered Clients, and provide different functionality to different Clients. A permanent identifier for the Client is needed for the Client

developer and the AS admin to ensure they are referring to the same Client. The Client ID was used in OAuth 2.0, and it served the same purpose.

8. *Why have both claims and authorizations?*

There are use cases for each that are independent: authenticating a user vs granting access to a resource. A request for an authorization returns an access token or handle, while a request for a claim returns the claim.

9. *Why specify HTTP/2 or later and TLS 1.3 or later for Client and AS communication in ?*Section 8

Knowing the AS supports HTTP/2 enables a Client to set up a connection faster. HTTP/2 will be more efficient when Clients have large numbers of access tokens and are frequently refreshing them at the AS as there will be less network traffic. Mandating TLS 1.3 similarly improves the performance and security of Clients and AS when setting up a TLS connection.

10. *Why do some of the JSON objects only have one child, such as the identifiers object in the user object in the initiation request?*

It is difficult to forecast future use cases. Having more resolution may mean the difference between a simple extension, and a convoluted extension.

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

The AS may use another AS to authenticate Users. The "iss" and "sub" combination is required to uniquely identify the User for any AS.

13. Acknowledgments

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

Additional thanks to Justin Richer for his strong critique of earlier

drafts.

14. IANA Considerations

[JOSE parameter for Authorization HTTP header]

TBD

15. Security Considerations

TBD

16. References

16.1. Normative References

[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>.

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

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

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

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

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[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>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[OIDC] Sakimora, 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>.

16.2. 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>.

[RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/info/rfc8252>.

[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>.

[RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <https://www.rfc-editor.org/info/rfc8628>.

[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>.

[RAR] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 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/>.

[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 Authorization", December 2019, <https://tools.ietf.org/html/draft-richer-transactional-authz-04>.

Appendix A. Document History

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

* Initial version

Appendix B. Comparison with OAuth 2.0 and OpenID Connect

Changed Features

The major differences between this protocol and OAuth 2.0 and OpenID Connect are:

* The Client uses a private key to authenticate in this protocol instead of the client secret in OAuth 2.0 and OpenID Connect.

* The Client initiates the protocol by making a signed request directly to the AS instead of redirecting the User to the AS.

* The Client does not receive any parameters from a redirection of the User back from the AS.

* Refreshing an access token requires creating a refresh token from a refresh handle, rather than an authenticated call with a refresh token.

* The Client can request identity claims to be returned independent of the ID Token. There is no UserInfo endpoint to query claims as there is in OpenID Connect.

Preserved Features

* This protocol reuses the OAuth 2.0 scopes, Client IDs, and access tokens of OAuth 2.0.

* This protocol reuses the Client IDs, Claims and ID Token of OpenID Connect.

* No change is required by the Client or the RS for existing APIs.

Author's Address

Dick Hardt (editor)
SignIn.Org
United States

Email: dick.hardt@gmail.com