G. Fernandez
Telefonica
F. Walter
A. Nennker
Deutsche Telekom AG
D. Tonge
Moneyhub
B. Campbell
Ping Identity
June 4, 2021

OpenID Connect Client-Initiated Backchannel Authentication Flow - Core 1.0 draft-04

openid-client-initiated-backchannel-authentication-core-04

Abstract

OpenID Connect Client Initiated Backchannel Authentication Flow is an authentication flow like OpenID Connect. However, unlike OpenID Connect, there is direct Relying Party to OpenID Provider communication without redirects through the user's browser. This specification has the concept of a Consumption Device (on which the user interacts with the Relying Party) and an Authentication Device (on which the user authenticates with the OpenID Provider and grants consent). This specification allows a Relying Party that has an identifier for a user to obtain tokens from the OpenID Provider. The user starts the flow with the Relying Party at the Consumption Device, but authenticates and grants consent on the Authentication Device.

Table of Contents

1. Introduction
   1.1. Requirements Notation and Conventions
2. Terminology
3. Overview
4. Registration and Discovery Metadata
5. Poll, Ping, and Push Modes
6. Example Use Cases
7. Backchannel Authentication Endpoint
   7.1. Authentication Request
      7.1.1. Signed Authentication Request
      7.1.2. User Code
   7.2. Authentication Request Validation
   7.3. Successful Authentication Request Acknowledgement
   7.4. Authentication Request Acknowledgment Validation
8. OpenID Provider Obtains End-User Consent/Authorization
9. Client Notification Endpoint
10. Getting the Authentication Result
   10.1. Token Request Using CIBA Grant Type
      10.1.1. Successful Token Response
   10.2. Ping Callback
   10.3. Push Callback
      10.3.1. Successful Token Delivery
11. Token Error Response
12. Push Error Payload
13. Authentication Error Response
14. Security Considerations
15. Privacy Considerations
16. IANA Considerations
   16.1. OAuth Authorization Server Metadata Registration
   16.2. OAuth Dynamic Client Registration Metadata Registration
   16.3. OAuth Parameters Registration
   16.4. JSON Web Token Claims
17. References
   17.1. Normative References
   17.2. Informative References

Appendix A. Acknowledgements

Appendix B. Notices

Appendix C. Document History

Authors' Addresses

1. Introduction

OpenID Connect allows Relying Parties (RP) to authenticate their users for clients of all types, including browser-based JavaScript and native mobile apps, to launch sign-in flows and receive verifiable assertions about the identity of signed-in users.

In all of these flows initiated by the RP, the end-user interaction from the consumption device is required and, they are based on HTTP redirection mechanisms. However, some use cases not covered by these flows have been raised, where the RP needs to be the initiator of the user authentication flow, and end-user interaction from the consumption device is not needed.

Client-Initiated Backchannel Authentication (CIBA) is a new authentication flow in which RPs, that can obtain a valid identifier for the user they want to authenticate, will be able to initiate an interaction flow to authenticate their users without having end-user interaction from the consumption device. The flow involves direct communication from the Client to the OpenID Provider without redirect through the user's browser (consumption device).

This specification does not change the semantics of the OpenID Connect Authentication flow. It introduces a new endpoint to which the authentication request is posted. It introduces a new asynchronous method for authentication result notification or delivery. It does not introduce new scope values nor does it change the semantics of standard OpenID Connect parameters.

As the user does not provide authentication credentials directly to the consumption device, supporting this flow requires the to OP have some mechanism of initiating user authentication out-of-band from the interaction with the consumption device.

1.1. Requirements Notation and Conventions

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

Throughout this document, values are quoted to indicate that they are to be taken literally. When using these values in protocol messages, the quotes MUST NOT be used as part of the value.

2. Terminology

This specification uses the terms "OpenID Provider (OP)" and "Relying Party (RP)" as defined by OpenID Connect Core. Furthermore, it uses the term "Client" as defined by OAuth 2.0. OAuth 2.0 Authorization Servers implementing OpenID Connect and CIBA are also referred to as OpenID Providers (OPs). OAuth 2.0 Clients using OpenID Connect and CIBA are also referred to as Relying Parties (RPs). This specification also uses the following terms:

Consumption Device (CD)
: The Consumption Device is the device that helps the user consume the service. In the CIBA use case, the user is not necessarily in control of the CD. For example, the CD may be in the control of an RP agent (e.g. at a bank teller) or might be a device controlled by the RP (e.g. a petrol pump).

Authentication Device (AD)
: The device on which the user will authenticate and authorize the request, often a smartphone.

3. Overview

Client-Initiated Backchannel Authentication (CIBA) enables a Client to initiate the authentication of an end-user through out-of-band mechanisms.

1. The Client shall make an "HTTP POST" request to the Backchannel Authentication Endpoint to ask for end-user authentication.
2. The OP will respond immediately with a unique identifier that identifies that authentication while it tries to authenticate the user in the background.
3. The Client will receive the ID Token, Access Token, and optionally Refresh Token through either the Poll, Ping, or Push modes, this choice MUST be established by the Client at registration time.

    Poll Mode
    : When configured in Poll mode, the Client will poll the token endpoint to get a response with the tokens.

    Ping Mode
    : When configured in Ping mode, the OP will send a request to a callback URI previously registered by the Client with the unique identifier returned from the Backchannel Authentication Endpoint. Upon receipt of the notification, the Client makes a request to the token endpoint to obtain the tokens.

Push Mode
: When configured in Push mode, the OP will send a request with the tokens to a callback URI previously registered by the Client.

This specification allows for some extensibility - profiles can be created that define additional parameters to be sent to, and received from, the Backchannel Authentication Endpoint.

4. Registration and Discovery Metadata

Grant Type
: This specification introduces the CIBA grant type (an extension grant type as defined by Section 4.5 of OAuth 2.0) with the value: `urn:openid:params:grant-type:ciba`

OpenID Provider Metadata
: The following authorization server metadata parameters are introduced by this specification for OPs publishing their support of the CIBA flow and details thereof.

  - `backchannel_token_delivery_modes_supported`: REQUIRED. JSON array containing one or more of the following values: `poll`, `ping`, and `push`.
  - `backchannel_authentication_endpoint`: REQUIRED. URL of the OP's Backchannel Authentication Endpoint as defined in Section 7.
  - `backchannel_authentication_request_signing_alg_values_supported`: OPTIONAL. JSON array containing a list of the JWS signing algorithms (alg values) supported by the OP for signed authentication requests, which are described in Section 7.1.1. If omitted, signed authentication requests are not supported by the OP.
  - `backchannel_user_code_parameter_supported`: OPTIONAL. Boolean value specifying whether the OP supports the use of the `user_code` parameter, with true indicating support. If omitted, the default value is false.

  The CIBA grant type is used in the `grant_types_supported` field of discovery metadata for OPs that support the ping or poll delivery modes.

  The supported client authentication methods and, when applicable, the associated JWS signing algorithms of the OP's Backchannel Authentication Endpoint are the same as those indicated by the `token_endpoint_auth_methods_supported` and `token_endpoint_auth_signing_alg_values_supported` metadata parameters respectively.

Client Metadata
: Clients registering to use CIBA MUST indicate a token delivery mode. When using the ping or poll mode, the Client MUST include the CIBA grant type in the "grant_types" field. When using the ping or push mode, the Client MUST register a client notification endpoint. Clients intending to send signed authentication requests MUST register the signature algorithm that will be used. The following parameters are introduced by this specification:

  - `backchannel_token_delivery_mode`: REQUIRED. One of the following values: `poll`, `ping`, or `push`.
  - `backchannel_client_notification_endpoint`: REQUIRED if the token delivery mode is set to `ping` or `push`. This is the endpoint to which the OP will post a notification after a successful or failed end-user authentication. It MUST be an HTTPS URL.
  - `backchannel_authentication_request_signing_alg`: OPTIONAL. The JWS algorithm `alg` value that the Client will use for signing authentication requests, as described in Section 7.1.1. When omitted, the Client will not send signed authentication requests.
  - `backchannel_user_code_parameter`: OPTIONAL. Boolean value specifying whether the Client supports the `user_code` parameter. If omitted, the default value is false. This parameter only applies when OP parameter `backchannel_user_code_parameter_supported` is true.

  The `token_endpoint_auth_method` indicates the registered authentication method for the client to use when making direct requests to the OP, including requests to both the token endpoint and the backchannel authentication endpoint.

Poll and Ping Modes with Pairwise Identifiers
: To use the Poll or Ping mode with Pairwise Pseudonymous Identifiers (PPIDs), the Client needs to register a URI that is of its ownership and use it during the authentication

process in a way that demonstrates that the URI belongs to it, which allows the OP to consider the host component of that URI as the Sector Identifier for the pairwise identifier calculation per Section 8.1 of OpenID Connect Core.

In OpenID Connect Core the `sector_identifier_uri` contains a document with a list of `redirect_uris` and the Sector Identifier is defined as either the host component of the `sector_identifier_uri` or if this is not provided then the host component of the `redirect_uri`.

In CIBA Poll and Ping modes the `jwks_uri` is used in place of the `redirect_uri`. In CIBA Push mode the `backchannel_client_notification_endpoint` is used in place of the `redirect_uri`. In situations where the PPID must be shared among multiple RPs, then a `sector_identifier_uri` can be registered. This specification extends the purpose of the `sector_identifier_uri` such that it can contain `jwks_uris` and `backchannel_client_notification_endpoints` as well as `redirect_uri`.

To support Pairwise Pseudonymous Identifiers in Ping and Poll modes, the RP must provide either a `sector_identifier_uri` or a `jwks_uri` at the registration phase when the `urn:openid:params:grant-type:ciba` grant type is registered. In that way, the OpenID Provider can use the host component of the `sector_identifier_uri` or `jwks_uri` as the Sector Identifier to generate the PPIDs for the Client.

When an OpenID Provider that supports PPIDs receives a dynamic registration request for a Client that indicates that it wishes to use the Poll or Ping CIBA modes, it MUST check if a valid `jwks_uri` is set when the `subject_type` is `pairwise`. If a `sector_identifier_uri` is explicitly provided, then the `jwks_uri` must be included in the list of URIs pointed to by the `sector_identifier_uri`.

But having registered a "jwks_uri" is not enough to use PPIDs, Client needs somehow to demonstrate that such "jwks_uri" belongs to it, which can be accomplished by proving possession of a private key corresponding to one of the public keys published at the "jwks_uri". Such proof can be demonstrated with signed authentication requests using the asymmetric keys provided by the "jwks_uri" or by authenticating to the OP using one of the following two mechanisms in conjunction with a key from its "jwks_uri":

1. Using the Self-Signed Certificate Mutual TLS OAuth Client Authentication Method as defined in section 2.2 of [RFC8705].
2. Using the private_key_jwt method as per section 9 Client Authentication of [OpenID.Core].

Push Mode with Pairwise Identifiers

When using the Push mode, the PPIDs will use the host component of the "backchannel_client_notification_endpoint" as the Sector Identifier. In case a "sector_identifier_uri" is explicitly provided, then the "backchannel_client_notification_endpoint" must be included in the list of URIs pointed to by the "sector_identifier_uri".

The following is a non-normative example from a dynamic registration request that contains the CIBA grant type as required and a "jwks_uri" (with line wraps within values for display purposes only).

```
POST /connect/register HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: server.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ...

{
    "application_type": "web",
    "client_name": "My Example",
    "logo_uri": "https://client.example.org/logo.png",
    "subject_type": "pairwise",
    "token_endpoint_auth_method": "private_key_jwt",
    "grant_types": ["urn:openid:params:grant-type:ciba"],
    "backchannel_token_delivery_mode": "poll",
    "jwks_uri": "https://client.example.org/my_public_keys.jwks",
    "contacts": ["ve7jtb@example.org", "mary@example.org"]
}
```

5. Poll, Ping, and Push Modes

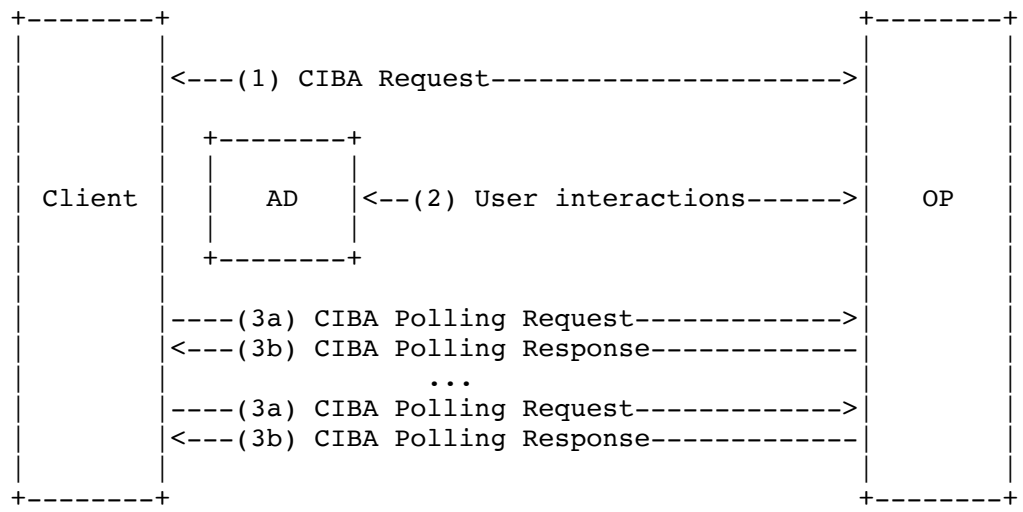
This specification allows the Client to get the authentication result in three ways: poll, ping, or push.

In the Poll mode, the authentication result is retrieved by the Client by polling the OP's token endpoint using the new grant type.

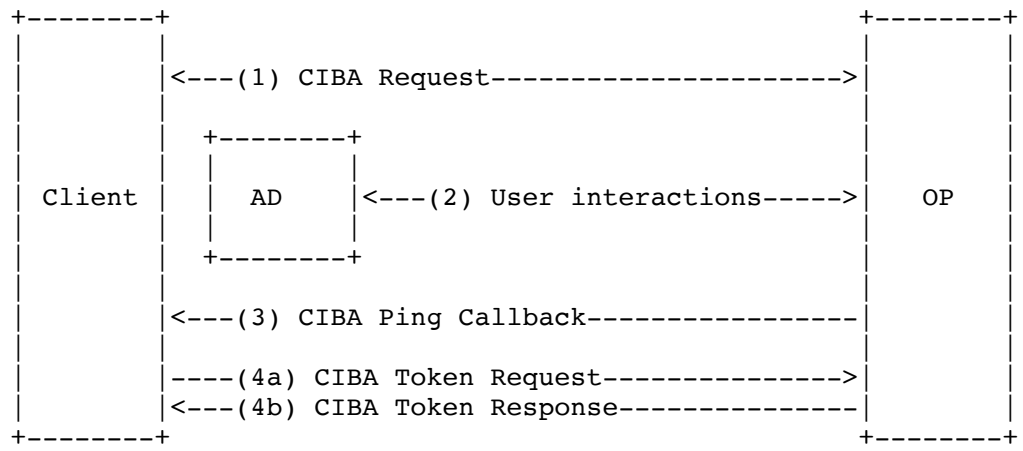
In the Ping mode, the OP will post the unique identifier of the authentication session to the Client, the Client will then retrieve the authentication result from the token endpoint using the new grant type.

In the Push mode, the OP will post the full authentication result to the Client.

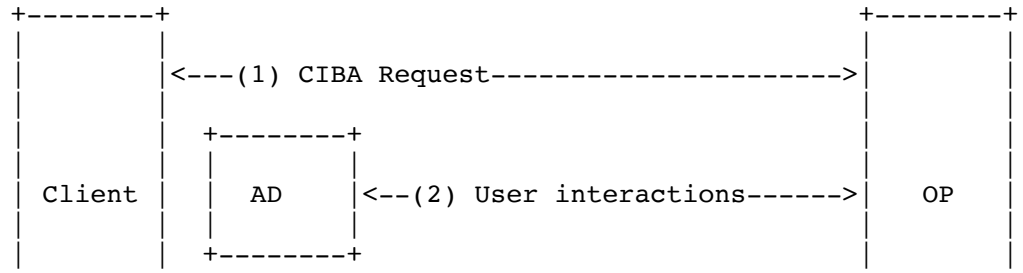
CIBA Poll Mode is illustrated in the following diagram:

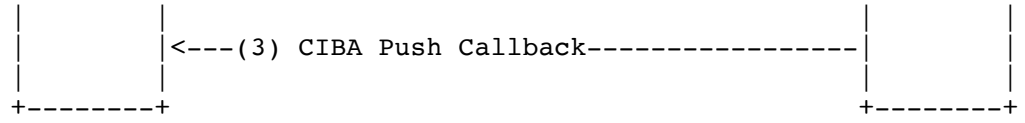


CIBA Ping Mode is illustrated in the following diagram:



CIBA Push Mode is illustrated in the following diagram:





Sender Constrained Tokens in Push Mode
: Any sender constraining method that relies on the Client presenting key material (i.e a certificate or a public key) at the Token Endpoint can be adapted to work with CIBA. For example, RFC8705: OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens describes how access tokens can be bound to a TLS client certificate. In the standard OAuth 2.0 flows, this binding occurs when the token is retrieved from the Token Endpoint. In CIBA Push Mode tokens are not retrieved from the Token Endpoint, however, tokens can be bound to the key material (e.g. client certificate) presented at the Backchannel Authentication Endpoint. OPs who wish to implement this functionality should keep some record of the key material that was presented at the Backchannel Authentication Endpoint (e.g. the SHA-256 Thumbprint) when the CIBA flow is initiated. When issuing the tokens via Push Mode the OP should bind the tokens to the key material that was presented at the Backchannel Authentication Endpoint.

6. Example Use Cases

The following use cases are non-normative examples to illustrate the usage of this specification.

1. A call center agent wants to authenticate a caller. Using additional scopes like e.g. "profile" or "phone" the call center agent would get access to claims about the user like "phone_number" and "phone_number_verified".
2. A bank teller wants to authenticate a customer in a bank branch - using CIBA for authentication in a face-to-face scenario.
3. A user wants to use their smartphone to authorize a payment they are making at a point of sale terminal.

7. Backchannel Authentication Endpoint

The Backchannel Authentication Endpoint is used to initiate an out-of-band authentication of the end-user. This is done by sending an HTTP POST message directly from the Client to the OpenID Provider's Backchannel Authentication Endpoint, using a request defined in the following subsections.

Communication with the Backchannel Authentication Endpoint MUST utilize TLS. See Section 16.17 [OpenID.Core] for more information on using TLS.

7.1. Authentication Request

Client-Initiated Backchannel Authentication defines an authentication request that is requested directly from the Client to the OpenID Provider without going through the user's browser. The Client MUST send an authentication request to the OpenID Provider by building an "HTTP POST" request that will take to the OpenID Provider all the information needed to authenticate the user without asking them for their identifier.

The Client MUST authenticate to the Backchannel Authentication Endpoint using the authentication method registered for its client_id, such as the authentication methods from Section 9 of [OpenID.Core] or authentication methods defined by extension in other specifications. Note that there's some potential ambiguity around the appropriate audience value to use when JWT client assertion based authentication is employed. To address that ambiguity the Issuer Identifier of the OP SHOULD be used as the value of the audience. To facilitate interoperability, the OP MUST accept its Issuer Identifier, Token Endpoint URL, or Backchannel Authentication Endpoint URL as values that identify it as an intended audience.

An authentication request is composed of the following parameters and MAY contain additional parameters defined by extension or profile:

scope
: REQUIRED. The scope of the access request as described by Section 3.3 of [RFC6749]. OpenID Connect implements authentication as an extension to OAuth 2.0 by including the

`openid` scope value in the authorization requests. Consistent with that, CIBA authentication requests MUST therefore contain the `openid` scope value. The behavior when the `openid` scope value is not present is left unspecified by this document, thus allowing for the potential definition of the behavior in a non-OpenID context, such as an OAuth authorization flow. Other scope values MAY be present, including but not limited to the `profile`, `email`, `address`, and `phone` scope values from Section 5.4 of [OpenID.Core].

client_notification_token
: REQUIRED if the Client is registered to use Ping or Push modes. It is a bearer token provided by the Client that will be used by the OpenID Provider to authenticate the callback request to the Client. The length of the token MUST NOT exceed 1024 characters and it MUST conform to the syntax for Bearer credentials as defined in Section 2.1 of [RFC6750]. Clients MUST ensure that it contains sufficient entropy (a minimum of 128 bits while 160 bits is recommended) to make brute force guessing or forgery of a valid token computationally infeasible - the means of achieving this are implementation-specific, with possible approaches including secure pseudorandom number generation or cryptographically secured self-contained tokens.

acr_values
: OPTIONAL. Requested Authentication Context Class Reference values. A space-separated string that specifies the acr values that the OpenID Provider is being requested to use for processing this Authentication Request, with the values appearing in order of preference. The actual means of authenticating the end-user, however, are ultimately at the discretion of the OP, and the Authentication Context Class satisfied by the authentication performed is returned as the `acr` Claim Value of the ID Token. When the `acr_values` parameter is present in the authentication request, it is highly RECOMMENDED that the resulting ID Token contains an `acr` Claim.

login_hint_token
: OPTIONAL. A token containing information identifying the end-user for whom authentication is being requested. The particular details and security requirements for the `login_hint_token` as well as how the end-user is identified by its content are deployment or profile specific.

id_token_hint
: OPTIONAL. An ID Token previously issued to the Client by the OpenID Provider being passed back as a hint to identify the end-user for whom authentication is being requested. If the ID Token received by the Client from the OP was asymmetrically encrypted, to use it as an id_token_hint, the client MUST decrypt the encrypted ID Token to extract the signed ID Token contained in it.

login_hint
: OPTIONAL. A hint to the OpenID Provider regarding the end-user for whom authentication is being requested. The value may contain an email address, phone number, account number, subject identifier, username, etc., which identifies the end-user to the OP. The value may be directly collected from the user by the Client before requesting authentication at the OP, for example, but may also be obtained by other means.

binding_message
: OPTIONAL. A human-readable identifier or message intended to be displayed on both the consumption device and the authentication device to interlock them together for the transaction by way of a visual cue for the end-user. This interlocking message enables the end-user to ensure that the action taken on the authentication device is related to the request initiated by the consumption device. The value SHOULD contain something that enables the end-user to reliably discern that the transaction is related across the consumption device and the authentication device, such as a random value of reasonable entropy (e.g. a transactional approval code). Because the various devices involved may have limited display abilities and the message is intending for visual inspection by the end-user, the `binding_message` value SHOULD be relatively short and use a limited set of plain text characters. The `invalid_binding_message` defined in Section 13 is used in the case that it is necessary to inform the Client that the provided `binding_message` is unacceptable.

user_code
: OPTIONAL. A secret code, such as a password or pin, that is known only to the user but verifiable by the OP. The code is used to authorize sending an authentication request to the user's authentication device. This parameter should only be present if the client registration parameter `backchannel_user_code_parameter` indicates support for the user code.

requested_expiry
: OPTIONAL. A positive integer allowing the client to request the `expires_in` value for the `auth_req_id` the server will return. The server MAY use this value to influence the lifetime of the authentication request and is encouraged to do so where it will improve the user experience, for example by terminating the authentication when as it knows the client is no longer interested in the result.

Because in the CIBA flow, the OP does not have an interaction with the end-user through the consumption device, it is REQUIRED that the Client provides one (and only one) of the hints specified above in the authentication request, that is "login_hint_token", "id_token_hint" or "login_hint".

An authentication request is made using the HTTP `POST` method with the aforementioned parameters in the `application/x-www-form-urlencoded` format and a character encoding of UTF-8 in the HTTP request entity-body. When applicable, additional parameters required by the given client authentication method are also included (e.g. JWT assertion-based client authentication uses `client_assertion` and `client_assertion_type` while Mutual TLS client authentication uses `client_id`).

The following is a non-normative example of an authentication request (with line wraps within values for display purposes only):

```
POST /bc-authorize HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

scope=openid%20email%20example-scope&
client_notification_token=8d67dc78-7faa-4d41-aabd-67707b374255&
binding_message=W4SCT&
login_hint_token=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.ey
JzdWJfaWQiOnsiZm9ybWF0IjoicGhvbmUiLCJwaG9uZSI6IisxMzMwMjgxODAwN
CJ9fQ.GSqxJsFbIyojdfMBDv3MOyAplCViVkwQWzthCWuu9_gnKIqECZilwANt1
HfIh3x3JFjaEq-5MZ_B3qeb11NAvg&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJ
pc3MiOiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0dHB
zOi8vc2VydmVyLmV4YW1wbGUuY29tIiwianRpIjoiYmRjLVhzX3NmLTNZTW80RlN
6SUoyUSIsImlhdCI6MTUzNzgxOTQ4NiwiZXhwIjoxNTM3ODE5Nzc3fQ.Ybr8mg_3
E2OptOSsA8rnelYO_y1L-yFaF_j1iemM3ntB61_GN3APe5cl_-5a6cvGlP154XAK
7fL-GaZSdnd9kg
```

7.1.1. Signed Authentication Request

A signed authentication request is made by encoding all of the authentication request parameters as claims of a signed JWT with each parameter name as the claim name and its value as a JSON string. An exception to this is `requested_expiry`, which may be sent as either a JSON string or a JSON number, the OP must accept either type. An extension or profile may define additional authentication request parameters, these may be defined to be any JSON type.

The JWT MUST contain all of the authentication request parameters. The JWT MUST be secured with an asymmetric signature and follow the guidance from Section 10.1 of [OpenID.Core] regarding asymmetric signatures. The JWT MUST also contain the following [RFC7519] registered claims:

aud
: The Audience claim MUST contain the value of the Issuer Identifier for the OP, which identifies the Authorization Server as an intended audience.

iss
: The Issuer claim MUST be the `client_id` of the OAuth Client.

exp
: An expiration time that limits the validity lifetime of the signed authentication request.

iat
: The time at which the signed authentication request was created.

nbf
: The time before which the signed authentication request is unacceptable.

jti
: A unique identifier for the signed authentication request.

The signed authentication request JWT is passed as an `application/x-www-form-urlencoded` HTTP request parameter with the name `request`. Authentication request parameters MUST NOT be present outside of the JWT, in particular, they MUST NOT appear as HTTP request parameters. Additional HTTP request parameters as required by the given client authentication method, however, MUST be included as `application/x-www-form-urlencoded` parameters (e.g. Mutual TLS client authentication uses `client_id` while JWT assertion-based client authentication uses `client_assertion` and `client_assertion_type`).

For example, a signed authentication request using the same authentication request parameters and values as the example from the previous section would look like the following (with line wraps within values for display purposes only):

```
POST /bc-authorize HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

request=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJpc3MiOiJz
NkJoZFJrcXQzIiwiYXVkIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJl
eHAiOjE1Mzc4MjAwODYsImlhdCI6MTUzNzgxOTQ4NiwibmJmIjoxNTM3ODE4ODg2
LCJqdGkiOiI0TFRDcUFDQzJFU0M1QldDbk4zajU4RW5BIiwic2NvcGUiOiJvcGVu
aWQgZW1haWwgZXhhbXBsZS1zY29wZSIsImNsaWVudF9ub3RpZmljYXRpb25fdG9r
ZW4iOiI4ZDY3ZGM3OC03ZmFhLTRkNDEtYWFiZC02NzcwN2IzNzQyNTUiLCJiaW5k
aW5nX21lc3NhZ2UiOiJXNFNDVCIsImxvZ2luX2hpbnRfdG9rZW4iOiJleUpyYVdR
aU9pSnNkR0ZqWlhOaWR5SXNJbUZzWnlJNklrVlRNalUySW4wLmV5SnpkV0pmYVdR
aU9uc2labTl5YldGMElqb2ljR2h2Ym1VaUxDSndhRzl1WlNJNklpc3hNek13TWpn
eE9EQXdOQ0o5ZlEuR1NxeEpzRmJJeW9qZGZNQkR2M01PeUFwbENWaVZrd1FXenRo
Q1d1dTlfZ25LSXFFQ1ppbHdBTnQxSGZJaDN4M0pGamFFcS01TVpfQjNxZWIxMU5B
dmcifQ.ELJvZ2RfBl05bq7nx7pXhagzL9R75mUwO-yZScB1aT3mp480fCQ5KjRVD
womMMjiMKUI4sx8VrPgAZuTfsNSvA&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJ
pc3MiOiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0dHB
zOi8vc2VydmVyLmV4YW1wbGUuY29tIiwianRpIjoiY2NfMVhzc3NmLTJpOG8yZ1B
6SUprMSIsImlhdCI6MTUzNzgxOTQ4NiwiZXhwIjoxNTM3ODE5Nzc3fQ.PWb_VMzU
IbD_aaO5xYpygnAlhRIjzoc6kxg4NixDuD1DVpkKVSBbBweqgbDLV-awkDtuWnyF
yUpHqg83AUV5TA
```

Where the following is the JWT payload (with line wraps and added whitespace for display purposes only):

```
{
 "iss": "s6BhdRkqt3",
 "aud": "https://server.example.com",
 "exp": 1537820086,
 "iat": 1537819486,
 "nbf": 1537818886,
 "jti": "4LTCqACC2ESC5BWCnN3j58EnA",
 "scope": "openid email example-scope",
 "client_notification_token": "8d67dc78-7faa-4d41-aabd-67707b374255",
 "binding_message": "W4SCT",
 "login_hint_token": "eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.
   eyJzdWJfaWQiOnsiZm9ybWF0IjoicGhvbmUiLCJwaG9uZSI6IisxMzMwMjgxODA
   wNCJ9fQ.GSqxJsFbIyojdfMBDv3MOyAplCViVkwQWzthCWuu9_gnKIqECZilwAN
   t1HfIh3x3JFjaEq-5MZ_B3qeb11NAvg"
}
```

Note that encrypted JWT authentication requests are not supported.

7.1.2. User Code

CIBA supports the optional use of "user codes" as a mechanism to prevent unsolicited authentication requests from appearing on a user's authentication device.

A "user code" is a secret known to the user, but which is not the user's password at the OP. When an OP supports this feature then Clients are required to ask the user for their "user code" in order to start a CIBA flow. This prevents malicious Clients or malicious end-users from starting unsolicited CIBA flows for a particular user for whom they know the login_hint or equivalent.

It is optional for the OP to implement User Code functionality. If the OP implements user code functionality then it may allow 1) clients without user code and 2) users without user code.

Typically clients that establish a security context with the user prior to sending a CIBA request should be allowed without the User Code mechanism (e.g. web applications that use CIBA for step-up authentication or call center applications where a caller ID is used to identify the user). In addition, clients who don't use static user identifiers as login hints should be allowed without requiring the User Code mechanism.

The OP declares support for user code with the provider metadata parameter `backchannel_user_code_parameter_supported`.

The Client registration parameter `backchannel_user_code_parameter` specifies if support for user code is required from the Client.

A client may detect users who require a user code from the authentication request error code. For example, a client may first attempt an authentication request without a user code and only prompt for a user code if it receives the error code `missing_user_code`.

The Client MUST NOT store the user code, but should rather request it from the user for each CIBA flow.

Registering a user code for a user is not in the scope of this specification. Examples include a facility provided by the authentication device or another service provided by the OP. OPs should provide a method for the user to change the user code.

7.2. Authentication Request Validation

The OpenID Provider MUST validate the request received as follows:

1. Authenticate the Client per the authentication method registered or configured for its client_id. It is RECOMMENDED that Clients not send shared secrets in the Authentication Request but rather that public-key cryptography be used.
2. If the authentication request is signed, validate the JWT sent with the `request` parameter, which includes verifying the signature and ensuring that the JWT is valid in all other respects per [RFC7519].
3. Validate all the authentication request parameters. In the event the request contains more than one of the hints specified in Authentication Request, the OpenID Provider MUST return an "invalid_request" error response as per Section 13.
4. The OpenID Provider MUST process the hint provided to determine if the hint is valid and if it corresponds to a valid user. The type, issuer (where applicable) and maximum age (where applicable) of a hint that an OP accepts should be communicated to Clients. How the OP validates hints and informs Clients of its hint requirements is out-of-scope of this specification.
5. If the hint is not valid or if the OP is not able to determine the user then an error should be returned to the Client as per Section Authentication Error Response.
6. The OpenID Provider MUST verify that all the REQUIRED parameters are present and their usage conforms to this specification.

OpenID Providers MUST ignore unrecognized request parameters.

If the OpenID Provider encounters any error, it MUST return an error response, per Section 13.

7.3. Successful Authentication Request Acknowledgement

If the Authentication Request is validated as per Section Authentication Request Validation, the OpenID Provider will return an HTTP 200 OK response to the Client to indicate that the authentication request has been accepted and is going to be processed. The body of this response will contain:

auth_req_id
: REQUIRED. This is a unique identifier to identify the authentication request made by the Client. It MUST contain sufficient entropy (a minimum of 128 bits while 160 bits is recommended) to make brute force guessing or forgery of a valid `auth_req_id` computationally infeasible - the means of achieving this are implementation-specific, with possible approaches including secure pseudorandom number generation or cryptographically secured self-contained tokens. The OpenID Provider MUST restrict the characters used to 'A'-'Z', 'a'-'z', '0'-'9', '.', '-' and '_', to reduce the chance of the client incorrectly decoding or re-encoding the `auth_req_id`; this character set was chosen to allow the server to use unpadded base64url if it wishes. The identifier MUST be treated as opaque by the client.

expires_in
: REQUIRED. A JSON number with a positive integer value indicating the expiration time of the "auth_req_id" in seconds since the authentication request was received. A Client calling the token endpoint with an expired `auth_req_id` will receive an error, see Token Error Response.

interval
: OPTIONAL. A JSON number with a positive integer value indicating the minimum amount of time in seconds that the Client MUST wait between polling requests to the token endpoint. This parameter will only be present if the Client is registered to use the Poll or Ping modes. If no value is provided, clients MUST use 5 as the default value.

Clients MUST ignore unrecognized response parameters.

The following is a non-normative example of an authentication response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "auth_req_id": "1c266114-a1be-4252-8ad1-04986c5b9ac1",
  "expires_in": 120,
  "interval": 2
}
```

7.4. Authentication Request Acknowledgment Validation

If the Client receives an HTTP 200 OK, it MUST validate that all the required parameters are received.

The Client MUST keep the `auth_req_id` in order to validate the callbacks received in Ping and Push modes or to use when making a token request in Poll and Ping modes.

The Client should store the expiration time in order to clean up authentication requests for which no Ping Callback or Push Callback is received.

8. OpenID Provider Obtains End-User Consent/Authorization

After the OP has validated the Authentication Request, the OP identifies the user and chooses an appropriate channel to authenticate the user and authorize the request, in line with the Client's requests regarding acr_values. Typically this involves the OP initiating an interactive session on the user's authentication device.

Once the end-user is authenticated, the OpenID Provider MUST obtain an authorization decision as described in Section 3.1.2.4 of [OpenID.Core].

When using the Client Initiated Backchannel Authentication flow, there is no interactive dialogue between the OpenID Provider and the end-user through the user's consumption device. There might be an agent of the Client involved who transfers the binding_message to the user.

9. Client Notification Endpoint

The Client Notification Endpoint is set by the Client during Registration and Discovery Metadata. It is the endpoint the OP will call after a successful or failed end-user authentication.

It MUST be an HTTPS URL and Communication with the Client Notification Endpoint MUST utilize TLS. See Section 16.17 [OpenID.Core] for more information on using TLS.

When the Client is configured in Ping mode, the endpoint receives a notification from the OP that an Authentication Result is ready to be retrieved from the Token Endpoint.

When the Client is configured in Push mode, the endpoint receives the Authentication Result (an ID Token, an Access Token, and optionally a Refresh Token or in the event that the user did not grant authorization, an error).

Requests to the Client Notification Endpoint must be authenticated using a "bearer token" created by the Client and sent to the OP in the Authentication request as the value of the parameter "client_notification_token".

10. Getting the Authentication Result

The token delivery mode for the Client (Poll, Ping, or Push) is determined at registration time.

A Client can only register a single token delivery method and the OP MUST only deliver the Authentication Result to the Client through the registered mode.

10.1. Token Request Using CIBA Grant Type

If the Client is registered to use Poll or Ping modes, the Client will retrieve the Authentication Result from the token endpoint.

The Client MUST be authenticated as specified in Section 9 of [OpenID.Core].

If the Client is registered to use the Poll mode, then the Client polls the token endpoint at a reasonable interval, which MUST NOT be more frequent than the minimum interval provided by the OpenID Provider via the "interval" parameter (if provided).

For polling requests, the OP may implement long polling, where the OP responds to the token request only when the authentication result has become available or a timeout has occurred. A timeout of 30 seconds is recommended for long polling requests, as specified in Section 5.5 of RFC6202, Best Practices for Long Polling.

Clients should be prepared to wait at least 30 seconds for a response when using polling mode. The OP should not take longer than 30 seconds to respond to a request, even when using long polling.

The polling interval is measured from the instant when a polling request is sent. To implement long polling the OP may respond slower than interval. A Client MUST NOT send two overlapping requests with the same `auth_req_id`. Rather the client must always wait until receiving the response to the previous request before sending out the next request. If the interval has passed when the response is received, then the client may immediately send out the next request.

An OP in a meltdown type situation may return an HTTP 503 with a Retry-After response header as described in Section 7.1.3 of HTTP/1.1. Clients should respect such headers and only retry

the token request after the time indicated in the header.

Here are some illustrations of how a Client should behave based on different OP responses, assuming a default interval of 5s:

Long Polling
: The Client makes a token request and the OP returns an `authorization_pending` error after 30s. In this case, the Client can immediately make the next token request.

Standard Polling
: The Client makes a token request and the OP returns an `authorization_pending` error after 2s. In this case, the Client must wait for 3s before making the next request.

OP responds slower than 30s
: The Client makes a token request and the OP doesn't return any response within 30s. In this case, the Client may cancel the request and make a new request.

If the Client is registered to use the Ping mode, then when the Client receives a notification to its Client Notification Endpoint containing an auth_req_id that is verified against a client_notification_token, it must call the token endpoint to retrieve the authentication result.

NOTE: A Client configured in Ping mode may also poll the token endpoint. The OpenID Provider must treat such a Client as if it had been registered to use the Poll mode.

The Client makes an "HTTP POST" request to the token endpoint by sending the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

grant_type
: REQUIRED. Value MUST be `urn:openid:params:grant-type:ciba`

auth_req_id
: REQUIRED. It is the unique identifier to identify the authentication request (transaction) made by the Client. The OP MUST check whether the auth_req_id was issued to this Client in response to an Authentication Request. Otherwise, an error MUST be returned.

The following is a non-normative example of a token request (with line wraps within values for display purposes only).

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aopenid%3Aparams%3Agrant-type%3Aciba&
auth_req_id=1c266114-a1be-4252-8ad1-04986c5b9ac1&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.ey
Jpc3MiOiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0d
HBzOi8vc2VydmVyLmV4YW1wbGUuY29tL3Rva2VuIiwianRpIjoiLV9wMTZqNkhj
aVhvMzE3aHZaMzEyYyIsImlhdCI6MTUzNzgxOTQ5MSwiZXhwIjoxNTM3ODE5Nzg
yfQ.BjaEoqZb-81gE5zz4UYwNpC3QVSeX5XhH176vg35zjkbq3Zmv_UpHB2ZugR
Va344WchTQVpaSSShLbvha4yziA
```

10.1.1. Successful Token Response

After receiving and validating a valid and authorized Token Request from the Client and when the end-user associated with the supplied `auth_req_id` has been authenticated and has authorized the request, the OpenID Provider returns a successful response as specified in Section 3.1.3.3 of [OpenID.Core]. Once redeemed for a successful token response, the `auth_req_id` value that was used is no longer valid. If the end-user associated with the supplied `auth_req_id` has not been authenticated or has not authorized the request, an error response must be sent as defined in Token Error Response.

The following is a non-normative example of a successful token response (with line wraps within values for display purposes only).

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
 "access_token": "G5kXH2wHvUra0sHlDy1iTkDJgsgUO1bN",
 "token_type": "Bearer",
 "refresh_token": "4bwc0ESC_IAhflf-ACC_vjD_ltc11ne-8gFPfA2Kx16",
 "expires_in": 120,
 "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjE2NzcyNiJ9.eyJpc3MiOiJo
   dHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsInN1YiI6IjI0ODI4OTc2MTAwMSIs
   ImF1ZCI6InM2QmhkUmtxdDMiLCJlbWFpbCI6ImphbmVkb2VAZXhhbXBsZS5jb20i
   LCJleHAiOjE1Mzc4MTk4MDMsImlhdCI6MTUzNzgxOTUwM30.aVq83mdy72ddIFVJ
   LjlNBX-5JHbjmwK-Sn9Mir-blesfYMceIOw6u4GOrO_ZroDnnbJXNKWAg_dxVynv
   MHnk3uJc46feaRIL4zfHf6Anbf5_TbgMaVO8iczD16A5gNjSD7yenT5fslrrW-NU
   _vtmi0s1puoM4EmSaPXCR19vRJyWuStJiRHK5yc3BtBlQ2xwxH1iNP49rGAQe_LH
   fW1G74NY5DaPv-V23JXDNEIUTY-jT-NbbtNHAxnhNPyn8kcO2WOoeIwANO9BfLF1
   EFWtjGPPMj6kDVrikec47yK86HArGvsIIwk1uExynJIv_tgZGE0eZI7MtVb2UlCw
   DQrVlg"
}
```

10.2. Ping Callback

If the Client is registered in Ping mode, the OpenID Provider will send an HTTP POST request to the Client Notification Endpoint after a successful or failed end-user authentication. A Ping callback does not need to be sent for an expired `auth_req_id` because the Client is already aware of the transaction lifetime via the `expires_in` authentication request acknowledgment parameter.

In this mode, the OP sends the `client_notification_token` as a bearer token in the Authorization header field and sends only the auth_req_id in the body of the request. The request uses the application/json media type.

The following is a non-normative example of a Ping callback sent as an HTTP POST request to the Client's Notification Endpoint (with line wraps within values for display purposes only).

```
POST /cb HTTP/1.1
Host: client.example.com
Authorization: Bearer 8d67dc78-7faa-4d41-aabd-67707b374255
Content-Type: application/json

{
 "auth_req_id": "1c266114-a1be-4252-8ad1-04986c5b9ac1"
}
```

The Client MUST verify the `client_notification_token` used to authenticate the request is valid and is associated with the `auth_req_id` received in the Ping callback. If the bearer token is not valid, the Client SHOULD return an HTTP 401 Unauthorized response.

For valid requests, the Client Notification Endpoint SHOULD respond with an HTTP 204 No Content. The OP SHOULD also accept responses with HTTP 200 OK and any body in the response SHOULD be ignored.

The Client MUST NOT return an HTTP 3xx code. The OP MUST NOT follow redirects.

How the OP handles HTTP error codes in the ranges of 4xx and 5xx is out-of-scope of this specification. Administrative action is likely to be needed in these cases.

For valid requests, the Client can now use the received `auth_req_id` to make a Token Request using the CIBA Grant Type to the Token Endpoint as described in Token Request Using CIBA Grant Type.

Clients MUST ignore unrecognized request parameters.

10.3. Push Callback

10.3.1. Successful Token Delivery

If the Client is registered in Push mode and the user is well authenticated and has authorized the request, the OpenID Provider delivers a payload that includes an ID Token, an Access Token, and optionally a Refresh Token to the Client Notification Endpoint.

Error conditions associated with the authentication request are delivered to the Client by sending a Push Error Payload to the Client Notification Endpoint.

The Push Callback uses the parameters defined for a successful token response in Section 4.1.4 of OAuth 2.0 and Section 3.1.3.3 of [OpenID.Core]. In addition, a new parameter "auth_req_id" is included in the payload. This is the authentication request identifier as defined in Successful Authentication Response.

The Push Callback uses the application/json media type.

In order to bind together the ID Token, the Access Token, and the auth_req_id, the OP MUST include the hash value of the Access Token and the auth_req_id within the ID Token using the `at_hash` and `urn:openid:params:jwt:claim:auth_req_id` claims respectively. In case a Refresh Token is sent to the Client, the hash value of it MUST also be added to the ID token using the `urn:openid:params:jwt:claim:rt_hash` claim. Section 3.1.3.6 of [OpenID.Core] shows how to calculate the hash value of the access_token for `at_hash`, the same method can also be applied to calculate the Refresh Token hash value. Note that these claims are only required in Push mode.

The following is a non-normative example of a Push Callback sent as an HTTP POST request to the Client's notification endpoint (with line wraps within values for display purposes only). The request is authenticated through a bearer token that is the value of the "client_notification_token" provided by the Client in the Authentication Request.

```
POST /cb HTTP/1.1
Host: client.example.com
Authorization: Bearer 8d67dc78-7faa-4d41-aabd-67707b374255
Content-Type: application/json

{
 "auth_req_id": "1c266114-a1be-4252-8ad1-04986c5b9ac1",
 "access_token": "G5kXH2wHvUra0sHlDy1iTkDJgsgUO1bN",
 "token_type": "Bearer",
 "refresh_token": "4bwc0ESC_IAhflf-ACC_vjD_ltc11ne-8gFPfA2Kx16",
 "expires_in": 120,
 "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjE2NzcyNiJ9.eyJpc3MiOiJ
   odHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsInN1YiI6IjI0ODI4OTc2MTAwMS
   IsImF1ZCI6InM2QmhkUmtxdDMiLCJlbWFpbCI6ImphbmVkb2VAZXhhbXBsZS5jb
   20iLCJleHAiOjE1Mzc4MTk4MDMsImlhdCI6MTUzNzgxOTUwMywiYXRfaGFzaCI6
   Ild0MGtWRlhNYWNxdm5IZXlVMDAwMXciLCJ1cm46b3BlbmlkOnBhcmFtczpqd3Q
   6Y2xhaW06cnRfaGFzaCI6InNIYWhDdVNwWENSZzVta0REdnZyNHciLCJ1cm46b3
   BlbmlkOnBhcmFtczpqd3Q6Y2xhaW06YXV0aF9yZXFfaWQiOiIxYzI2NjExNC1hM
   WJlLTQyNTItOGFkMS0wNDk4NmM1YjlhYzEifQ.SGB5_a8E7GjwtoYrkFyqOhLK6
   L8-Wh1nLeREwWj30gNYOZW_ZB2mOeQ5yiXqeKJeNpDPssGUrNo-3N-CqNrbmVCb
   XYTwmNB7IvwE6ZPRcfxFV22oou-NS4-3rEa2ghG44Fi9D9fVURwxrRqgyezeD3H
   HVIFUnCxHUou3OOpj6aOgDqKI4Xl2xJ0-kKAxNR8LljUp64OHgoS-UO3qyfOwIk
   IAR7o4OTK_3Oy78rJNT0Y0RebAWyA81UDCSf_gWVBp-EUTI5CdZ1_odYhwB9OWD
   W1A22Sf6rmjhMHGbQW4A9Z822yiZZveuT_AFZ2hi7yNp8iFPZ8fgPQJ5pPpjA7u
   dg"
}
```

The Client MUST verify the client_notification_token used to authenticate the request is valid and is associated with the auth_req_id received in the Push Callback. If the bearer token is not valid the Client SHOULD return an HTTP 401 Unauthorized response.

The Client MUST validate the ID Token, which acts as a detached signature, as per Section 3.1.3.7 of [OpenID.Core].

The Client MUST ensure that the value of the `urn:openid:params:jwt:claim:auth_req_id` claim in the ID Token matches the auth_req_id in the request.

The Client MUST validate the access token received using the at_hash in the ID Token as per Section 3.2.2.9 of [OpenID.Core]. If a refresh token is present, the Client MUST validate it using the `urn:openid:params:jwt:claim:rt_hash` in the ID Token in a similar manner as the access token is validated.

For valid requests, the Client Notification Endpoint SHOULD respond with an HTTP 204 No Content. The OP SHOULD also accept HTTP 200 OK and any content in the response SHOULD be ignored.

The Client MUST NOT return an HTTP 3xx code. The OP MUST NOT follow redirects.

How the OP handles HTTP error codes in the ranges of 4xx and 5xx is out-of-scope of this specification. Administrative action is likely to be needed in these cases.

The following is a non-normative example of a base64url decoded ID Token sent to the client notification endpoint:

```
{
  "iss": "https://server.example.com",
  "sub": "248289761001",
  "aud": "s6BhdRkqt3",
  "email": "janedoe@example.com",
  "exp": 1537819803,
  "iat": 1537819503,
  "at_hash": "Wt0kVFXMacqvnHeyU0001w",
  "urn:openid:params:jwt:claim:rt_hash": "sHahCuSpXCRg5mkDDvvr4w",
  "urn:openid:params:jwt:claim:auth_req_id":
    "1c266114-a1be-4252-8ad1-04986c5b9ac1"
}
```

Clients MUST ignore unrecognized request parameters.

11. Token Error Response

If the Token Request is invalid or unauthorized, the OpenID Provider constructs an error response according to Section 3.1.3.4 Token Error Response of [OpenID.Core]. In addition to the error codes defined in Section 5.2 of [RFC6749], the following error codes defined in the OAuth Device Flow are also applicable:

authorization_pending
: The authorization request is still pending as the end-user hasn't yet been authenticated.

slow_down
: A variant of "authorization_pending", the authorization request is still pending and polling should continue, but the interval MUST be increased by at least 5 seconds for this and all subsequent requests.

expired_token
: The auth_req_id has expired. The Client will need to make a new Authentication Request.

access_denied
: The end-user denied the authorization request.

If the auth_req_id is invalid or was issued to another Client, an `invalid_grant` error MUST be returned as described in Section 5.2 of [RFC6749].

If a Client continually polls quicker than the `interval`, the OP may return an `invalid_request` error.

If a Client receives an `invalid_request` error it must not make any further requests for the same `auth_req_id`.

If the Client is registered to use the Push Mode then it MUST NOT call the Token Endpoint with the CIBA Grant Type and the following error is returned.

unauthorized_client
: The Client is not authorized as it is configured in Push Mode

Note, when a Client receives a 4xx response code with a JSON payload then it should inspect the payload in order to determine the error rather than relying on the HTTP status code alone.

12. Push Error Payload

When Clients are configured to use the Push token delivery mode they can receive error payloads at their Client Notification Endpoint. These errors will be sent using the application/json media type with the following three parameters:

error_description
: OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error_description" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E

error
: REQUIRED. A single ASCII error code from one present in the list below.

auth_req_id
: REQUIRED. The authentication request identifier.

Error codes applicable to the push error payload:

access_denied
: The end-user denied the authorization request.

expired_token
: The auth_req_id has expired. The Client will need to make a new Authentication Request. OpenID Providers are not required to send this error, but Clients SHOULD support receiving this error.

transaction_failed
: The OpenID Provider encountered an unexpected condition that prevented it from successfully completing the transaction. This general case error code can be used to inform the Client that the CIBA transaction was unsuccessful for reasons other than those explicitly defined by `access_denied` and `expired_token`.

13. Authentication Error Response

An Authentication Error Response is returned directly from the Backchannel Authentication Endpoint in response to the Authentication Request sent by the Client. The applicable error codes are detailed below (some of which are repurposed from OAuth 2.0 Sections 4.1.2.1 and 5.2).

Authentication Error Responses are sent in the same format as Token Error Responses, i.e. the HTTP response body uses the application/json media type with the following parameters:

error
: REQUIRED. A single ASCII error code from one present in the list below.

error_description
: OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error_description" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

error_uri

OPTIONAL. A URI identifying a human-readable web page with information about the error to provide the client developer with additional information. Values for the "error_uri" parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.

List of authentication error codes associated with HTTP Errors.

HTTP 400 Bad Request

invalid_request
: The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, contains more than one of the hints, or is otherwise malformed.

invalid_scope
: The requested scope is invalid, unknown, or malformed.

expired_login_hint_token
: The login_hint_token provided in the authentication request is not valid because it has expired.

unknown_user_id
: The OpenID Provider is not able to identify which end-user the Client wishes to be authenticated by means of the hint provided in the request (login_hint_token, id_token_hint, or login_hint).

unauthorized_client
: The Client is not authorized to use this authentication flow.

missing_user_code
: User code is required but was missing from the request.

invalid_user_code
: The user code was invalid.

invalid_binding_message
: The binding message is invalid or unacceptable for use in the context of the given request.

HTTP 401 Unauthorized

invalid_client
: Client authentication failed (e.g., invalid client credentials, unknown client, no client authentication included, or unsupported authentication method).

HTTP 403 Forbidden

access_denied
: The resource owner or OpenID Provider denied the request. Note that as the authentication error response is received prior to any user interaction, such an error would only be received if a resource owner or OpenID Provider had made a decision to deny a certain type of request or requests from a certain type of client. The mechanism for such a decision to be made is outside the scope of this specification.

The following is a non-normative example from an authentication error response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "error": "unauthorized_client",
  "error_description":
    "The client 'client.example.org' is not allowed to use CIBA."
}
```

14. Security Considerations

The login_hint_token SHOULD be digitally signed by the issuer. This ensures the authenticity of the data and reduces the threat of an injection attack. The signature allows the OP to authenticate and authorize the sender of the hint and prevent the collecting of user identifiers by rogue Clients.

The OP SHOULD ensure that the "backchannel_client_notification_endpoint" configured at registration time is in the administrative authority of the Client. Otherwise, the OP would deliver authentication results to the wrong Client. How this check is done is outside the scope of this specification.

An id_token_hint cannot be validated using standard JWT processing rules because the token is being used in a context (sent from the Client back to the OP) that is different than that for which it was originally issued (typically a short-lived token issued by the OP intended to convey claims about the authentication of an end-user to the Client). An expired ID Token therefore could still be considered valid as an id_token_hint so an OP should, for some reasonable period, accept id_token_hints with an expiration time that has passed. The OP should ensure that it is the issuer of the token and that the Client presenting the id_token_hint is listed in the audience claim. The OP should verify the signature to ensure that the token was, in fact, issued by it and hasn't been modified since. Note that due to key rotation, however, the OP may not necessarily have access to the key used to sign the token, so the length of time an ID Token is considered a valid hint will be likely to be limited by the OP's key rotation interval and retention period. Given these restrictions, implementers may consider not verifying the signature at all and only accepting ID Tokens with pairwise subject identifiers as hints. The OP could then validate that the Client authenticated at the Backchannel Authentication Endpoint was issued the pairwise subject identifier (or shares a Sector Identifier with the Client who was issued the pairwise subject identifier).

This specification defines a method of token delivery, "Push Callback", which differs from standard OAuth 2.0 flows. Most OAuth 2.0 and OpenID Connect profiles require the Client to authenticate at the token endpoint in order to retrieve access tokens, ID Tokens, and refresh tokens. However, with the CIBA Push mode, tokens are delivered directly to the Client at its Client Notification Endpoint. Implementers should ensure that appropriate security controls are in place for this endpoint and its registration with the OP. CIBA requires that a hash of the access token, a hash of the refresh token, and the auth_req_id itself are included in the ID Token when the Push mode is used. This allows the Client to verify that the values in the Push Callback have not been tampered with. Implementers using push tokens should also consider issuing sender-constrained access tokens to mitigate the risk of the tokens being intercepted.

In some scenarios, it may be appropriate for the RP to pass metadata to the OP about the context of the session it has with the user. For example, the RP could send the geolocation of the consumption device and the OP could verify that against the geolocation of the authentication device. This specification does not require such metadata to be sent, nor does it define the method by which such metadata would be conveyed.

15. Privacy Considerations

This flow requires the Client to obtain an identifier for the end-user. When this identifier is a static global identifier, such as a phone number or email address, there are clear privacy implications. However this flow does not require the use of such identifiers and in deployments with higher privacy requirements, alternative identifiers could be used, such as:

ID Token containing a pairwise identifier
: CIBA supports the use of ID Tokens as an `id_token_hint` in the authentication request. If the OP has previously issued an ID Token to the Client that contains a pairwise identifier and no personally identifiable information, then a CIBA flow can be initialized without the RP asking the user for a static identifier. This ID Token may have been obtained by the Client via a standard front-channel redirect flow.

Single-use user identifier, transferred from the AD to the CD
: The OP could generate a single-use identifier that could be transferred from the Authentication Device to the Consumption Device at the start of the flow. For example, the user could authenticate on the AD and request an identifier for a CIBA flow. This could be displayed as a QR code, the user could scan this QR code at the CD. The RP would then decode the identifier and embed it in a `login_hint_token` which it would use to initiate a CIBA flow. The OP would then gain consent from the end-user for the access requested by the RP.

Discovery Service
: For some ecosystems, the RP may be able to send the user to a discovery service and receive back an encrypted `login_hint_token` which it can use in the authentication request. The OP would decrypt the token, identify the user and continue the CIBA flow.

16. IANA Considerations

16.1. OAuth Authorization Server Metadata Registration

This specification requests registration of the following values in the IANA "OAuth Authorization Server Metadata" registry of [IANA.OAuth.Parameters] established by [RFC8414].

- Metadata Name: `backchannel_token_delivery_modes_supported`
- Metadata Description: Supported CIBA authentication result delivery modes
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

- Metadata Name: `backchannel_authentication_endpoint`
- Metadata Description: CIBA Backchannel Authentication Endpoint
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

- Metadata Name: `backchannel_authentication_request_signing_alg_values_supported`
- Metadata Description: JSON array containing a list of the JWS signing algorithms supported for validation of signed CIBA authentication requests
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

- Metadata Name: `backchannel_user_code_parameter_supported`
- Metadata Description: Indicates whether the OP supports the use of the CIBA `user_code` parameter.
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

16.2. OAuth Dynamic Client Registration Metadata Registration

This specification requests registration of the following client metadata definitions in the IANA "OAuth Dynamic Client Registration Metadata" registry of [IANA.OAuth.Parameters] established by [RFC7591]:

- Client Metadata Name: `backchannel_token_delivery_mode`
- Client Metadata Description: CIBA authentication result delivery mode
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

- Client Metadata Name: `backchannel_client_notification_endpoint`
- Client Metadata Description: Endpoint to which a notification will be sent after a CIBA end-user authentication event
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

- Client Metadata Name: `backchannel_authentication_request_signing_alg`
- Client Metadata Description: The JWS algorithm that the client will use to sign CIBA authentication requests
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

- Client Metadata Name: `backchannel_user_code_parameter`
- Client Metadata Description: Indicates whether the Client must support the CIBA `user_code` parameter.
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

16.3. OAuth Parameters Registration

This specification requests registration of the following value in the IANA "OAuth Parameters" registry of [IANA.OAuth.Parameters] established by [RFC6749].

- Parameter name: auth_req_id
- Parameter usage location: token response
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification document(s): Section 7.3 of this document

16.4. JSON Web Token Claims

This specification makes no request of IANA with respect to the "JSON Web Token Claims" registry at [IANA.JWT] established by [RFC7519]. The Public Claim Names `urn:openid:params:jwt:claim:auth_req_id` and `urn:openid:params:jwt:claim:rt_hash` were used for the auth_req_id and refresh token hash respectively in Section 10.3.1 in order to avoid further congestion of the registry with application-specific claims that are unlikely to be of general applicability. For a Signed Authentication Request, where the authentication request parameters are encoded as claims of the request JWT, the context of use is sufficiently constrained so as to safely consider those to be Private Claim Names.

17. References

17.1. Normative References

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and C. Mortimore, "OpenID Connect Core 1.0", November 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012.

[RFC7519] Jones, M., Bradley, J. and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N. and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020.

17.2. Informative References

[IANA.JWT] IANA, "JSON Web Token (JWT)"

[IANA.OAuth.Parameters] IANA, "OAuth Parameters"

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014.

[RFC7591] Richer, J., Jones, M., Bradley, J., Machulak, M. and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015.

[RFC8414] Jones, M., Sakimura, N. and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018.

[RFC8628] Denniss, W., Bradley, J., Jones, M. and H. Tschofenig, "OAuth 2.0

Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019.

Appendix A. Acknowledgements

The following have contributed to the development of this specification.

John Bradley, Ralph Bragg, Geoff Graham, Joseph Heenan, Bjorn Hjelm, Takahiko Kawasaki, Torsten Lodderstedt, James Manger, Charles Marais, Nat Sakimura, and Petteri Stenius.

Appendix B. Notices

Appendix C. Document History

[[To be removed from the final specification]]

-01

- Initial draft

-02

- Second draft
- Authentication Request Section: Improving the definition of `client_req_id`
- Successful Authentication Request Acknowledgement: a. `auth_req_id`: to explain that it will not be present in token when using Polling mode b. interval: fixing a misleading description
- Token Request Using Polling Mechanism: a. Fixing a misleading description about the inclusion of `client_notification_endpoint` in the authentication request. It does not make sense since Notification or Polling mode is defined at the registration time and `client_notification_endpoint`. is not sent in the authentication request anymore. b. `auth_req_id`: fixing misleading description.
- Changing Successful Token Polling to Successful Token Polling Response
- Improving descriptions in Successful Token Polling Response and Successful Token Notification
- `expires_in` parameter from Successful Authentication Request Acknowledgement refers to the `auth_req_id` that will be considered overdue to make new polling requests after that time.

- New `unknown_auth_req_id` and `expired_token` erros in Token Error Response
- Authentication Error Response section is defined and incorporates two new errors: `unknown_user_id` when OP cannot figure out the user to be authenticated by means of the hint and `expired_token` to indicate that the login_hint_token or id_token_hint provided is expired.
- Changing `client_req_id`to `client_notification_token`
- New overview section

-03

- Following the lead of user questioning remove the term Relying Party in most places.

-04

- The Client MUST be authenticated.
- The Client SHOULD use a signed OpenID Connect Request object.
- 3) The OP MUST support signed OpenID Connect Requests objects and if the validation of the signature fails the request MUST fail. If alg == none another method of Client authentication MUST be used
- Added text regarding Pairwise Identifiers.
- Added SIM Applet Authenticator example to illustrate the CIBA flow with an specific authenticator

-05

- If the OP uses Pairwise Identifiers then registering a sector_identifier_uri at registration time is mandatory.

-06

- If the OP uses Pairwise Identifiers and the Polling Mechanism to retrieve the token, it is necessary to provide a "jwks_uri" at the registration phase and it is MANDATORY for the Client to authenticate the token endpoint using one of this two mechanisms: Using Mutual TLS Self-Signed Certificate method as defined in section 2.2 of draft-ietf-oauth-mtls or using the private_key_jwt method as per the section 9 Client Authentication of the [OpenID.Core]
- at_hash, urn:openid:params:jwt:claim:rt_hash and urn:openid:params:jwt:claim:auth_req_id values have been added within the id_token to bind it with the access_token, refresh_token and the auth_req_id in the push mode.
- Change acr_values to be optional in the authentication request and define the parameter inline.
- Clarify that error responses from the Backchannel Authentication Endpoint are as defined in the Authentication Error Response section.
- Be clear that refresh tokens are optional.
- Introduce a third mode for getting the tokens so now there's 1) Poll, 2) Ping and 3) Push.
- Add AS/OP metadata parameter for the Backchannel Authentication Endpoint.
- More clearly define the authentication request format and client authentication to the Backchannel Authentication Endpoint.
- Change client_notification_endpoint to backchannel_client_notification_endpoint.

-OIDC-CIBA-CORE-01

- Changed name to OpenID Connect Client Initiated Backchannel Authentication Flow - Core
- Removed SimApplet Example
- Changed Privacy Considerations to be less MNO specific and explain the different identifiers that can be used.

-OIDC-CIBA-CORE-02

- Update the grant type URI to be more generic and not include MODRNA.
- Add an invalid_binding_message error code that can be used on the Authentication Error Response if the binding_message is unacceptable.
- Add the requested_expiry Authentication Request parameter.
- Add a transaction_failed error code to Push Error Payload as a catch-all.
- Be explicit that expires_in and interval of the Authentication Request Acknowledgement are positive integers represented as JSON numbers.

-OIDC-CIBA-CORE-03

- Explicitly state the expectations for the audience value when JWT assertion-based client authentication is used at the Backchannel Authentication Endpoint.
- Clarify that only an asymmetrically encrypted id_token_hint needs to be decrypted.
- Example changes: use public key crypto for client auth, make the expires_in values more reasonable, remove userinfo_encrypted_response_alg RSA1_5 use that wasn't needed anyway, update the login hint token to use sub_id rather than subject now that https://tools.ietf.org/html/draft-ietf-secevent-subject-identifiers-04 has a "sub_id" JWT Claim definition, change signed request and hint to use ECDSA so they're a little smaller.
- Define a default value of 5 seconds for interval.
- More clearly define Consumption Device and Authentication Device in the intro.
- Added a security consideration around context metadata.
- Better describe the use of long polling.
- Improve text around the PPID mechanism and the guidance around verification of ownership of keys at jwks_uri.
- Clarify that the auth_req_id value is not valid after successful redemption.
- Update some references.
- Add a note instructing clients to look error bodies rather than relying on the HTTP status code alone.
- Add character restrictions for auth_req_id values.
- Allow requested expiry to be sent as a number or string and state that extensions profiles may define additional authentication request parameters as any JSON type.
- Add guidance around clients who poll to quickly.
- Editorial changes to better introduce the user code mechanism.

-OIDC-CIBA-CORE-04

- Update OAuth MTLS draft reference to RFC 8705.
- Note in security considerations that the push mode doesn't work with the standardized PoP token issuance methods.
- Update login_hint_token example to align with change in subject identifiers draft where "subject_type" is now "format".
- Move 6749 to normative
- Clarify that a Ping notification isn't necessarily sent if the whole transaction expires

Authors' Addresses

Gonzalo Fernandez Rodriguez
Telefonica I+D
EMail: gonzalo.fernandezrodriguez@telefonica.com

Florian Walter
Deutsche Telekom AG
EMail: F.Walter@telekom.de

Axel Nennker
Deutsche Telekom AG
EMail: axel.nennker@telekom.de

Dave Tonge
Moneyhub
EMail: dave.tonge@moneyhub.com

Brian Campbell
Ping Identity
EMail: bcampbell@pingidentity.com