

Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

T. Looker
MATTR
P. Bastian
Bundesdruckerei
C. Bormann
SPRIND
2 March 2026

OAuth 2.0 Attestation-Based Client Authentication
draft-ietf-oauth-attestation-based-client-auth-08

Abstract

This specification defines an extension to the OAuth 2.0 protocol [RFC6749] that enables a client instance to include a key-bound attestation when interacting with an Authorization Server or Resource Server. This mechanism allows a client instance to prove its authenticity verified by a client attester without revealing its target audience to that attester. It may also serve as a mechanism for client authentication as per OAuth 2.0.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at <https://oauth-wg.github.io/draft-ietf-oauth-attestation-based-client-auth/draft-ietf-oauth-attestation-based-client-auth.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-oauth-attestation-based-client-auth/>.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (<mailto:oauth@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/oauth/>. Subscribe at <https://www.ietf.org/mailman/listinfo/oauth/>.

Source for this draft and an issue tracker can be found at <https://github.com/oauth-wg/draft-ietf-oauth-attestation-based-client-auth>.

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 3 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Conventions and Definitions	5
3. Terminology	5
4. Relation to RATS	6
5. Client Attestation Format	6
5.1. Client Attestation JWT	6
5.2. Client Attestation PoP JWT	8
6. Client Attestation using a Header based syntax	9
6.1. Client Attestation HTTP Headers	9
6.2. Validating HTTP requests featuring client attestations .	10
6.3. Client Attestation as an OAuth Client Authentication .	11
6.4. Client Attestation as an additional security signal . .	12
7. Challenge Retrieval	14
7.1. Providing Challenges on Previous Responses	15
8. Verification and Processing	16
9. Authorization Server Metadata	17
10. Implementation Considerations	18
10.1. Reuse of a Client Attestation JWT	18
10.2. Refresh token binding	18
10.3. Web Server Default Maximum HTTP Header Sizes	18

10.4.	Rotation of Client Instance Key	18
10.5.	Replay Attack Detection	19
11.	Privacy Considerations	19
11.1.	Client Instance Tracking Across Authorization Servers	19
12.	Security Considerations	19
12.1.	Replay Attacks	20
12.2.	Client Attestation Protection	21
13.	IANA Considerations	21
13.1.	OAuth Parameters Registration	21
13.2.	OAuth Extensions Error Registration	22
13.3.	OAuth Authorization Server Metadata Registration	23
13.4.	Registration of attest_jwt_client_auth Token Endpoint Authentication Method	23
13.5.	HTTP Field Name Registration	24
14.	References	24
14.1.	Normative References	24
14.2.	Informative References	26
Appendix A.	Document History	26
Acknowledgments	30
Authors' Addresses	30

1. Introduction

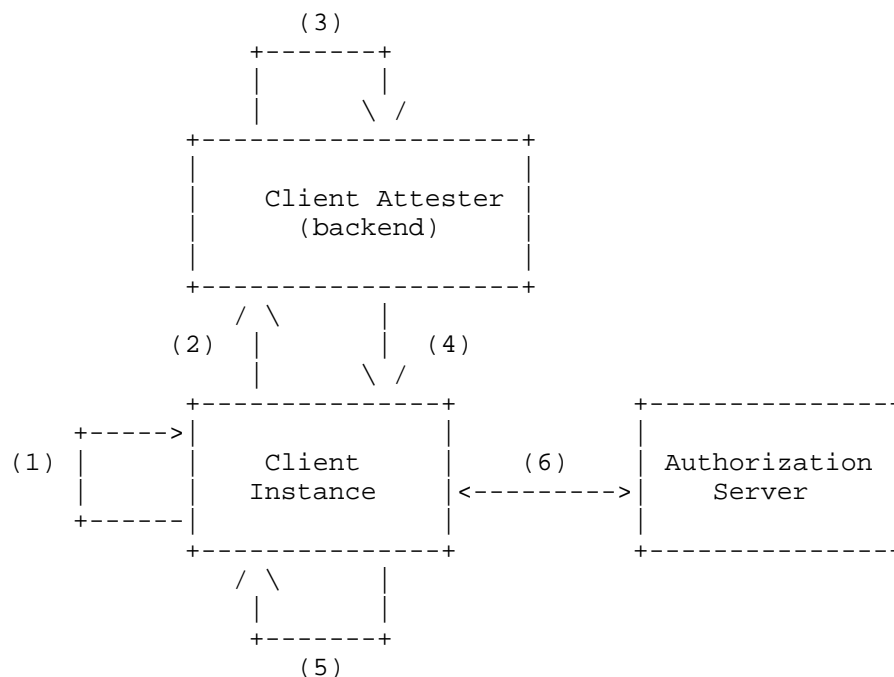
Traditional OAuth security concepts perform client authentication through a backend channel. In ecosystems such as the Issuer-Holder-Verifier model used in [RFC9901], this model raises privacy concerns, as it would enable the backend to recognize which Holder (i.e. client) interacts with which Issuer (i.e. Authorization Server) and potentially furthermore see the credentials being issued. This specification establishes a mechanism for a backend-attested client authentication through a frontend channel to address these issues.

Additionally, this approach acknowledges the evolving landscape of OAuth 2 deployments, where the ability for mobile native apps to authenticate securely and reliably has become increasingly important. Leveraging platform mechanisms to validate a client instance, such as mobile native apps, enables secure authentication that would otherwise be difficult with traditional OAuth client authentication methods. Transforming these platform-specific mechanisms into a common format as described in this specification abstracts this complexity to minimize the efforts for the Authorization Server.

This primary purpose of this specification is the authentication of a client instance enabled through the client backend attesting to it. The client backend may also attest further technical properties about the hardware and software of the client instance.

The client is considered a confidential OAuth 2 client type according to section 2.1 of [RFC6749]. The mechanism described in this document may either serve as a standalone OAuth 2 client authentication mechanism or as an additional, supportive security mechanism beside an existing OAuth 2 client authentication mechanism.

The following diagram depicts the overall architecture and protocol flow.



The following steps describe this OAuth flow:

(1) The Client Instance generates a key (Client Instance Key) and optional further attestations (that are out of scope) to prove its authenticity to the Client Attester.

(2) The Client Instance sends this data to the Client Attester in request for a Client Attestation JWT.

(3) The Client Attester validates the Client Instance Key and optional further data. It generates a signed Client Attestation JWT that is cryptographically bound to the Client Instance Key generated by the Client. Therefore, the attestation is bound to this particular Client Instance.

(4) The Client Attester responds to the Client Instance by sending the Client Attestation JWT.

(5) The Client Instance generates a Proof of Possession (PoP) with the Client Instance Key.

(6) The Client Instance sends both the Client Attestation JWT and the Client Attestation PoP JWT to the authorization server, e.g. within a token request. The authorization server validates the Client Attestation and thus authenticates the Client Instance.

Please note that the protocol details for steps (2) and (4), particularly how the Client Instance authenticates to the Client Attester, are beyond the scope of this specification. Furthermore, this specification is designed to be flexible and can be implemented even in scenarios where the client does not have a backend serving as a Client Attester. In such cases, each Client Instance is responsible for performing the functions typically handled by the Client Attester on its own.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

Client Attestation JWT: A JSON Web Token (JWT) generated by the Client Attester which is bound to a key managed by a Client Instance which can then be used by the instance for client authentication.

Client Attestation Proof of Possession (PoP) JWT: A Proof of Possession generated by the Client Instance using the key that the Client Attestation JWT is bound to.

Client Instance: A deployed instance of a piece of client software.

Client Instance Key: A cryptographic asymmetric key pair that is generated by the Client Instance where the public key of the key pair is provided to the Client Attester. This public key is then encapsulated within the Client Attestation JWT and is utilized to sign the Client Attestation Proof of Possession.

Client Attester: An entity that authenticates a Client Instance and

attests it by issuing a Client Attestation JWT.

Challenge: A String that is the input to a cryptographic challenge-response pattern. This is traditionally called a nonce within OAuth.

4. Relation to RATS

The Remote Attestation Procedures (RATS) architecture defined by [RFC9334] has some commonalities to this document. The flow specified in this specification relates to the "Passport Model" in RATS. However, while the RATS ecosystem gives explicit methods and values how the RATS Attester proves itself to the Verifier, this is deliberately out of scope for Attestation-Based Client Authentication. Additionally, the terminology between RATS and OAuth is different:

- * a RATS "Attester" relates to an OAuth "Client"
- * a RATS "Relying Party" relates to an OAuth "Authorization Server or Resource Server"
- * a RATS "Verifier" relates to the "Client Attester" defined in this specification
- * a RATS "Attestion Result" relates to the "Client Attestation JWT" defined by this specification
- * a RATS "Endorser", "Reference Value Provider", "Endorsement", "Evidence" and "Policies and Reference Values" are out of scope for this specification

5. Client Attestation Format

This draft introduces the concept of client attestations to the OAuth 2 protocol, using two JWTs: a Client Attestation and a Client Attestation Proof of Possession (PoP). The primary purpose of these JWTs is to authenticate the Client Instance. These JWTs are transmitted via HTTP headers in an HTTP request (as described in Section 6.1) from a Client Instance to an Authorization Server or Resource Server.

5.1. Client Attestation JWT

The Client Attestation MUST be encoded as a "JSON Web Token (JWT)" according to [RFC7519].

The following content applies to the JWT Header:

- * `typ`: REQUIRED. The JWT type MUST be `oauth-client-attestation+jwt`.

The following content applies to the JWT Claims Set:

- * `sub`: REQUIRED. The `sub` (subject) claim MUST specify `client_id` value of the OAuth Client.
- * `exp`: REQUIRED. The `exp` (expiration time) claim MUST specify the time at which the Client Attestation is considered expired by its issuer. The authorization server MUST reject any JWT with an expiration time that has passed, subject to allowable clock skew between systems.
- * `cnf`: REQUIRED. The `cnf` (confirmation) claim MUST specify a key conforming to [RFC7800] that is used by the Client Instance to generate the Client Attestation PoP JWT for client authentication with an authorization server. The key MUST be expressed using the "jwk" representation.
- * `iat`: OPTIONAL. The `iat` (issued at) claim MUST specify the time at which the Client Attestation was issued.

The following additional rules apply:

1. The JWT MAY contain other claims. All claims that are not understood by implementations MUST be ignored.
2. The JWT MUST be digitally signed or integrity protected with a Message Authentication Code (MAC). The authorization server MUST reject JWTs if signature or integrity protection validation fails.
3. The authorization server MUST reject a JWT that is not valid in all other respects per "JSON Web Token (JWT)" [RFC7519].

The following example is the decoded header and payload of a JWT meeting the processing rules as defined above.

```
{
  "typ": "oauth-client-attestation+jwt",
  "alg": "ES256",
  "kid": "11"
}
.
{
  "sub": "https://client.example.com",
  "iat": 1772487595,
  "exp": 2529866394,
  "cnf": {
    "jwk": {
      "kty": "EC",
      "use": "sig",
      "crv": "P-256",
      "x": "VcKvNBZ4IaBAYW3jxM4w3TJFVA7myeUGQyGt-g_yvpQ",
      "y": "f-E-hYE3TAWKwhVv9pej9NABs9SX9XsNO80x57jFTyU"
    }
  }
}
```

5.2. Client Attestation PoP JWT

The Client Attestation PoP MUST be encoded as a "JSON Web Token (JWT)" according to [RFC7519].

The following content applies to the JWT Header:

- * `typ`: REQUIRED. The JWT type MUST be `oauth-client-attestation-pop+jwt`.

The following content applies to the JWT Claims Set:

- * `aud`: REQUIRED. The `aud` (audience) claim MUST specify a value that identifies the authorization server as an intended audience. The [RFC8414] issuer identifier URL of the authorization server MUST be used as a value for an "aud" element to identify the authorization server as the intended audience of the JWT.
- * `jti`: REQUIRED. The `jti` (JWT identifier) claim MUST specify a unique identifier for the Client Attestation PoP. The authorization server can utilize the `jti` value for replay attack detection, see Section 12.1.
- * `iat`: REQUIRED. The `iat` (issued at) claim MUST specify the time at which the Client Attestation PoP was issued. Note that the authorization server may reject JWTs with an "iat" claim value that is unreasonably far in the past.

- * challenge: OPTIONAL. The challenge (challenge) claim MUST specify a String value that is provided by the authorization server for the client to include in the Client Attestation PoP JWT.

The following additional rules apply:

1. The JWT MAY contain other claims. All claims that are not understood by implementations MUST be ignored.
2. The JWT MUST be digitally signed using an asymmetric cryptographic algorithm. The authorization server MUST reject JWTs with an invalid signature.
3. The public key used to verify the JWT MUST be the key located in the "cnf" claim of the corresponding Client Attestation JWT.
4. The Authorization Server MUST reject a JWT that is not valid in all other respects per "JSON Web Token (JWT)" [RFC7519].

The following example is the decoded header and payload of a JWT meeting the processing rules as defined above.

```
{
  "typ": "oauth-client-attestation-pop+jwt",
  "alg": "ES256"
}
.
{
  "aud": "https://as.example.com",
  "jti": "d25d00ab-552b-46fc-ae19-98f440f25064",
  "challenge": "5c1a9e10-29ff-4c2b-ae73-57c0957c09c4"
}
```

6. Client Attestation using a Header based syntax

The following section defines how a Client Attestation can be provided in an HTTP request using HTTP headers.

6.1. Client Attestation HTTP Headers

When using headers to transfer the Client Attestation JWT and Client Attestation PoP JWT to an Authorization Server, they MUST be provided in an HTTP request using the following HTTP headers.

OAuth-Client-Attestation: A JWT that conforms to the structure and syntax as defined in Section 5.1

OAuth-Client-Attestation-PoP: A JWT that adheres to the structure

and syntax as defined in Section 5.2

The following is an example of the OAuth-Client-Attestation header.

```
OAuth-Client-Attestation: eyJ0eXAiOiJvYXV0aC1jbGllbnQtYXR0ZXN0YXRpb24rand0IiwiYWxnIjoirVMYNTYiLCJraWQiOiIxMSJ9.eyJzdWIiOiJodHRwczovL2NsaWVudC5leGFtcGxlLmNvbSIsImhhdCI6MTc3MjQ4NzU5NSwiZXhwIjoyNTI5ODY2Mzk0LCJjbmYiOiJ0eSI6IkdVdIiwidXNlIjoic2lnIiwiaWF0IjoiUC0yNTYiLCJ4IjoivMNLV5CWjRjYUJlbnVzZanhnNNHczVEpGVke3bXl1VUdReUd0LWdfexZwUSIsInkiOiJmLUUtaFlFMlRBV0t3aFZ2OXBlajlOQUJzOVNYOVhzTk84MHg1N2pGVHlVInl9fQ._TS4d-LAnRlwdN97wiVnl4z7C9gvm45IWr-BvGTzeZaHtZtgNZ88gvzroU3LElUPbgF4kWi_DF0RnKzsx5yu6A
```

The following is an example of the OAuth-Client-Attestation-PoP header.

```
OAuth-Client-Attestation-PoP: eyJhbGciOiJFUzI1NiIsInR5cCI6Im9hdXRoLWNsaWVudC1hdHRlc3RhdGlvbilwb3Arand0In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwianRpIjoizDI1ZDAwYWItNTUyYi00NmZjLWFlMTktOTNmNDQwZjI1MDY0Iiwibm9uY2UiOiI1YzFhOWUxMC0yOWZmLTRjMmItYWU3My01N2MwOTU3YzA5YzQifQ.U0uAUL60MXSf2qB3uWooltQanBMLa7OJ-pk_GsA_olrfJfrkUOyWpqsSbNH90ykVad-m6x5McrEnFgCqdkNfUA
```

Note that per [RFC9110] header field names are case-insensitive; so OAUTH-CLIENT-ATTESTATION, oauth-client-attestation, etc., are all valid and equivalent header field names. Case is significant in the header field value, however.

The OAuth-Client-Attestation and OAuth-Client-Attestation-PoP HTTP header field values uses the token68 syntax defined in Section 11.2 of [RFC9110] (repeated below for ease of reference).

```
OAuth-Client-Attestation      = token68
OAuth-Client-Attestation-PoP  = token68
token68                       = 1*( ALPHA / DIGIT / "-" / "." /
                                "_" / "~" / "+" / "/" ) * "="
```

It is RECOMMENDED that the authorization server validate the Client Attestation JWT prior to validating the Client Attestation PoP.

6.2. Validating HTTP requests featuring client attestations

To validate an HTTP request which contains the client attestation headers, the receiving server MUST ensure the following with regard to a received HTTP request:

1. There is precisely one OAuth-Client-Attestation HTTP request header field, where its value is a single well-formed JWT conforming to the syntax outlined in Section 5.1.
2. There is precisely one OAuth-Client-Attestation-PoP HTTP request header field, where its value is a single well-formed JWT conforming to the syntax outlined in Section 5.2.
3. The signature of the Client Attestation PoP JWT obtained from the OAuth-Client-Attestation-PoP HTTP header verifies with the Client Instance Key contained in the cnf claim of the Client Attestation JWT obtained from the OAuth-Client-Attestation HTTP header.

When validation errors specifically related to the use of client attestations are encountered the following additional error codes are defined for use in either Authorization Server authenticated endpoint error responses (as defined in Section 5.2 of [RFC6749]) or Resource Server error responses (as defined in Section 3 of [RFC6750]).

- * `use_attestation_challenge` MUST be used when the Client Attestation PoP JWT is not using an expected server-provided challenge. When used this error code MUST be accompanied by the OAuth-Client-Attestation-Challenge HTTP header field parameter (as described in Section 7.1).
- * `use_fresh_attestation` MUST be used when the Client Attestation JWT is deemed to be not fresh enough to be acceptable by the server.
- * `invalid_client_attestation` MAY be used in addition to the more general `invalid_client` error code as defined in [RFC6749] if the attestation or its proof of possession could not be successfully verified.

In the event of errors due to situations not described above, Authorization and Resource Servers MUST follow the guidance of [RFC6749] and [RFC6750] or their respective extensions of when to return suitable Error Responses.

6.3. Client Attestation as an OAuth Client Authentication

A Client Attestation may be used as an OAuth 2 Client Authentication mechanism as described in Section 2.3 of [RFC6749] towards an Authorization Server. In this case, the Authorization Server MUST perform all of the checks outlined in Section 6.2 for a received access token request or similar endpoints.

The following example demonstrates usage of the client attestation mechanism in an access token request (with extra line breaks for display purposes only):

6.4. Client Attestation as an additional security signal

The following example demonstrates usage of the client attestation mechanism in a PAR request as defined in [RFC9126] along side client secret (with extra line breaks for display purposes only):

```
response_type=code
&state=af0ifjsldkj
&client_id=s6BhdRkqt3
&client_secret=7Fjfp0ZBr1KtDRbnfVdmIw
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
&code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoEKlt8U
&code_challenge_method=S256&scope=account-information
```

The following example demonstrates usage of the client attestation mechanism at the Resource Server (with extra line breaks for display purposes only):

```
POST /api/users/list HTTP/1.1
Host: rs.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Bearer mF_9.B5f-4.1JqM
Accept: application/json
OAuth-Client-Attestation: eyJ0eXAiOiJvYXV0aC1jbGllbnQtYXR0ZXN0YXRpb24
rand0IiwiYWxnIjoirVMyNTYiLCJraWQiOiIxMSJ9.eyJzdWIiOiJodHRwczovL2NsaWV
udC5leGFtcGxlLmNvbSIsImhhdCI6MTc3MjQ4NzU5NSwiZXhwIjoyNTI5ODY2Mzk0LCJj
bmYiOiJ0eSI6IkdldiIiwidXNlIjoic2lnIiwiaWF0IjoiUC0yNTYiLCJ4I
joiVmNLVWk5CWJrJYUJBWVczanhNNHczVEpGVkE3bXl1VudReUd0LWdfeXZwUSIsInkiOi
JmLUUtaFlFMlRBV0t3aFZ2OXBlajlOQUJzOVNYOVhzTk84MHg1N2pGVHlVInl9fQ._TS4
d-LAnRlwdN97wiVnl4z7C9gvm45IWw-BvGTzeZaHtZtgNZ88gvzroU3LElUPbgF4kWi_D
FORnKzsx5yu6A
OAuth-Client-Attestation-PoP: eyJhbGciOiJFUzI1NiIsInR5cCI6Im9hdXRoLWN
saWVudC1hdHRlc3RhdGlvbilwb3Arand0In0.eyJhdWQiOiJodHRwczovL3JzLmV4YWlw
bGUuY29tIiwianRpIjoizDI1ZDAwYWItNTUyYi00NmZjLWFlMTktOThmNDQwZjI1MDY0I
iwibm9uY2UyOiIlYzFhOWUxMC0yOWZmLTRjMmItYWU3My01N2MwOTU3YzA5YzQifQ.gzk
0WkWsJNNx92gVMSp6jVpPDUvR0toYxLMYGMJMJOfm8mG2Otg0Nfm4PefOUpwBMNQIXSd
dW-cqJopljQaCQ
```

7. Challenge Retrieval

This section defines an optional mechanism that allows a Client to request a fresh Challenge from the Authorization Server to be included in the Client Attestation PoP JWT. This construct may be similar or equivalent to a nonce, see Section 3. The value of the challenge is opaque to the client.

An Authorization Server MAY offer a challenge endpoint for Clients to fetch Challenges in the context of this specification. If the Authorization Server supports metadata as defined in [RFC8414], it MUST signal support for the challenge endpoint by including the metadata entry `challenge_endpoint` containing the URL of the endpoint as its value. If the Authorization Server offers a challenge endpoint, the Client MUST retrieve a challenge and MUST use this challenge in the OAuth-Attestation-PoP as defined in Section 5.2.

A request for a Challenge is made by sending an HTTP POST request to the URL provided in the `challenge_endpoint` parameter of the Authorization Server metadata.

The following is a non-normative example of a request:

```
POST /as/challenge HTTP/1.1
Host: as.example.com
Accept: application/json
```

The Authorization Server provides a Challenge in the HTTP response with a 200 status code and the following parameters included in the message body of the HTTP response using the application/json media type:

- * `attestation_challenge`: REQUIRED if the authorization server supports Client Attestations and server provided challenges as described in this document. String containing a Challenge to be used in the OAuth-Attestation-PoP as defined in Section 5.2. The intention of this element not being required in other circumstances is to preserve the ability for the challenge endpoint to be used in other applications unrelated to client attestations.

The Authorization Server MUST make the response uncacheable by adding a Cache-Control header field including the value no-store. The Authorization Server MAY add additional challenges or data.

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

```
HTTP/1.1 200 OK
Host: as.example.com
Content-Type: application/json
Cache-Control: no-store

{
  "attestation_challenge": "AYjcyMzY3ZDhiNmJkNTZ"
}
```

7.1. Providing Challenges on Previous Responses

The Authorization Server MAY provide a fresh Challenge with any HTTP response using a HTTP header-based syntax. The HTTP header field parameter MUST be named "OAuth-Client-Attestation-Challenge" and contain the value of the Challenge. The Client MUST use this new Challenge for the next OAuth-Client-Attestation-PoP.

The following is a non-normative example of an Authorization Response containing a fresh Challenge:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
OAuth-Client-Attestation-Challenge: AYjcyMzY3ZDhiNmJkNTZ
```

```
{
  "access_token": "2YotnFZFEjrlzCsicMWpAA",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

8. Verification and Processing

Upon receiving a Client Attestation, the receiving server MUST ensure the following conditions and rules:

1. If the Client Attestation was received via Header based Syntax (as described in Section 6):
 - * The HTTP request contains exactly one field OAuth-Client-Attestation and one field OAuth-Client-Attestation-PoP.
 - * Both fields contain exactly one well-formed JWT.
2. The Client Attestation JWT contains all claims and header parameters as per Section 5.1.
3. The Client Attestation PoP JWT contains all claims and header parameters as per Section 5.2.
4. The alg JOSE Header Parameter for both JWTs indicates a registered asymmetric digital signature algorithm [IANA.JOSE.ALGS], is not none, is supported by the application, and is acceptable per local policy. The alg JOSE Header Parameter for the Client Attestation JWT can also be a registered Message Authentication Code (MAC).
5. The signature of the Client Attestation JWT verifies with the public key of a known and trusted Attester.
6. The key contained in the cnf claim of the Client Attestation JWT is not a private key.
7. The signature of the Client Attestation PoP JWT verifies with the public key contained in the cnf claim of the Client Attestation JWT.

8. If the server provided a challenge value to the client, the challenge claim is present in the Client Attestation PoP JWT and matches the server-provided challenge value.
9. The creation time of the Client Attestation PoP JWT as determined by either the iat claim or a server managed timestamp via the challenge claim, is within an acceptable window.
10. The audience claim in the Client Attestation PoP JWT is the issuer identifier URL of the authorization server as described in [RFC8414].
11. The Client Attestation JWT is fresh enough for the policies of the authorization server by checking the iat or exp claims.
12. Depending on the security requirements of the deployment, additional checks to guarantee replay protection for the Client Attestation PoP JWT might need to be applied (see Section 12.1 for more details).
13. If a client_id is provided in the request containing the Client Attestation, then this client_id matches the sub claim of the Client Attestation JWT.

9. Authorization Server Metadata

The Authorization Server SHOULD communicate support and requirement for authentication with Attestation-Based Client Authentication by using the value `attest_jwt_client_auth` in the `token_endpoint_auth_methods_supported` within its published metadata. The client SHOULD fetch and parse the Authorization Server metadata and recognize Attestation-Based Client Authentication as a client authentication mechanism if the given parameters are present.

The Authorization Server SHOULD communicate supported algorithms for client attestations by using `client_attestation_signing_alg_values_supported` and `client_attestation_pop_signing_alg_values_supported` within its published metadata. This enables the client to validate that its client attestation is understood by the Authorization Server prior to authentication. The client MAY try to get a new client attestation with different algorithms. The Authorization Server MUST include `client_attestation_signing_alg_values_supported` and `client_attestation_pop_signing_alg_values_supported` in its published metadata if the `token_endpoint_auth_methods_supported` includes `attest_jwt_client_auth`.

10. Implementation Considerations

10.1. Reuse of a Client Attestation JWT

Implementers should be aware that the design of this authentication mechanism deliberately allows for a Client Instance to re-use a single Client Attestation JWT in multiple interactions/requests with an Authorization Server, whilst producing a fresh Client Attestation PoP JWT. Client deployments should consider this when determining the validity period for issued Client Attestation JWTs as this ultimately controls how long a Client Instance can re-use a single Client Attestation JWT.

10.2. Refresh token binding

Authorization servers issuing a refresh token in response to a token request using the client attestation mechanism as defined by this draft MUST bind the refresh token to the Client Instance and its associated public key, and NOT just the client as specified in section 6 [RFC6749]. To prove this binding, the Client Instance MUST use the client attestation mechanism when refreshing an access token. The client MUST also use the same key that was present in the "cnf" claim of the client attestation that was used when the refresh token was issued.

10.3. Web Server Default Maximum HTTP Header Sizes

Because the Client Attestation and Client Attestation PoP are communicated using HTTP headers, implementers should consider that web servers may have a default maximum HTTP header size configured which could be too low to allow conveying a Client Attestation and or Client Attestation PoP in an HTTP request. It should be noted, that this limit is not given by the HTTP [RFC9112], but instead web server implementations commonly set a default maximum size for HTTP headers. As of 2024, typical limits for modern web servers configure maximum HTTP headers as 8 kB or more as a default.

10.4. Rotation of Client Instance Key

This specification does not provide a mechanism to rotate the Client Instance Key in the Client Attestation JWT's "cnf" claim. If the Client Instance needs to use a new Client Instance Key for any reason, then it MUST request a new Client Attestation JWT from its Client Attester.

10.5. Replay Attack Detection

Authorization Servers implementing measures to detect replay attacks as described in Section 12.1 require efficient data structures to manage large amounts of challenges for use cases with high volumes of transactions. To limit the size of the data structure, the Authorization Server should use a sliding window, allowing Client Attestation PoPs within a certain time window, in which the seen challenge or jti values are stored, but discarded afterwards. To ensure security, Client Attestation PoPs outside this time window MUST be rejected by the Authorization Server. The allowed window is determined by the iat of the Client Attestation PoP and the sliding window time duration chosen by the Authorization Server. These data structures need to:

- * search the data structure to validate whether a challenge from a Client Attestation PoP has been previously seen
- * insert the new challenges from the Client Attestation PoP if the search returned no result
- * delete the challenges after the Client Attestation PoP has passed the sliding time window

A trie (also called prefix tree), or a patricia trie (also called radix tree) is a RECOMMENDED data structures to implement such a mechanism.

11. Privacy Considerations

11.1. Client Instance Tracking Across Authorization Servers

Implementers should be aware that using the same client attestation across multiple authorization servers could result in correlation of the end user using the Client Instance through claim values (including the Client Instance Key in the cnf claim). Client deployments are therefore RECOMMENDED to use different Client Attestation JWTs with different Client Instance Keys across different authorization servers.

12. Security Considerations

The guidance provided by [RFC7519] and [RFC8725] applies.

12.1. Replay Attacks

An Authorization Server SHOULD implement measures to detect replay attacks by the Client Instance. In the context of this specification, this means to detect that an attacker is resending the same Client Attestation PoP JWT in multiple requests. The following options are RECOMMENDED for this client authentication method:

- * The Authorization Server manages a list of witnessed jti values of the Client Attestation PoP JWT for the time window of which the JWT would be considered valid. This sliding time window is based on the iat of the Client Attestation PoP and the duration chosen by the Authorization Server. If any Client Attestation PoP JWT would be replayed, the Authorization Server would recognize the jti value in the list and respond with an authentication error. Details how to implement such a data structure to maintain jti values is given in Section 10.5.
- * The Authorization Server provides a challenge as an OAuth-Client-Attestation-Challenge in the challenge endpoint to the Client Instance and the Client uses it as a challenge value in the Client Attestation PoP JWT. The Authorization Server may choose to:
 - manage a list of witnessed challenge values, similar to the previously described jti approach. Details how to implement such a data structure to maintain challenge values is given in Section 10.5. This guarantees stronger replay protection with a challenge chosen by the Authorization Server itself, at the potential cost of an additional round-trip.
 - use self-contained challenges while not storing the seen challenges. This approach scales well, while only guaranteeing freshness, but no replay protection within the limited time-window chosen by the Authorization Server.
- * The Authorization Server generates a challenge that is bound to the Client Instance's session, such that a specific challenge in the Client Attestation PoP JWT is expected and validated. The Authorization Server may either:
 - send the challenge as part of another previous response to the Client Instance of providing the challenge explicitly
 - reuse an existing artefact of the Client Instance's session, e.g. the authorization code. This MUST be communicated out-of-band between Authorization Server and Client.

Note that protocols that provide a challenge as part of a previous response should provide a clear indicator for clients when this feature is used. This makes it easier for client implementations to deal with proper state handling. This can be implicit by always mandating support for this feature or via some metadata that allows the client to detect support for this feature for a specific server.

Because clock skews between servers and clients may be large, Authorization Servers MAY limit Client Attestation PoP lifetimes by using server-provided challenge values containing the time at the server rather than comparing the client-supplied iat time to the time at the server. Challenges created in this way yield the same result even in the face of arbitrarily large clock skews.

In any case the Authorization Server SHOULD ensure the freshness of the Client Attestation PoP by checking either the iat claim or if present the server provided challenge, is within an acceptable time window.

The approach using a challenge explicitly provided by the Authorization Server gives stronger replay attack detection guarantees, however support by the Authorization Server is OPTIONAL to simplify mandatory implementation requirements. The jti value is mandatory and hence acts as a default fallback.

12.2. Client Attestation Protection

This specification allows both, digital signatures using asymmetric cryptography, and Message Authentication Codes (MAC) to be used to protect Client Attestation JWTs. Implementers should only use MACs to secure the integrity of Client Attestations JWTs if they fully understand the risks of MACs when compared to digital signatures and especially the requirements of their use-case scenarios. These use-cases typically represent deployments where the Client Attester and Authorization Server have a trust relationship and the possibility to securely exchange keys out of band or are the same entity and no other entity needs to verify the Client Attestations. We expect most deployments to use digital signatures for the protection of Client Attestations, and implementers SHOULD default to digital signatures if they are unsure.

13. IANA Considerations

13.1. OAuth Parameters Registration

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

- * Metadata Name: challenge_endpoint
- * Metadata Description: URL of the authorization servers challenge endpoint which is used to obtain a fresh challenge for usage in client authentication methods such as client attestation.
- * Change Controller: IETF
- * Reference: Section 7 of this specification

13.2. OAuth Extensions Error Registration

This specification requests registration of the following values in the IANA "OAuth Extensions Error Registry" registry of [IANA.OAuth.Params] established by [RFC6749].

- * Name: use_attestation_challenge
 - * Usage Location: token error response, resource access error response
 - * Protocol Extension: OAuth 2.0 Attestation-Based Client Authentication
 - * Change Controller: IETF
 - * Reference: Section 6.2 of this specification
-
- * Name: use_fresh_attestation
 - * Usage Location: token error response, resource access error response
 - * Protocol Extension: OAuth 2.0 Attestation-Based Client Authentication
 - * Change Controller: IETF
 - * Reference: this specification
-
- * Name: invalid_client_attestation
 - * Usage Location: token error response, resource access error response

- * Protocol Extension: OAuth 2.0 Attestation-Based Client Authentication
- * Change Controller: IETF
- * Reference: Section 6.2 of this specification

13.3. OAuth Authorization Server Metadata Registration

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

- * Metadata Name: client_attestation_signing_alg_values_supported
- * Metadata Description: JSON array containing a list of the JWS signing algorithms supported by the authorization server for the signature on the Client Attestation JWT.
- * Change Controller: IETF
- * Reference: Section 6.2 of this specification

- * Metadata Name: client_attestation_pop_signing_alg_values_supported
- * Metadata Description: JSON array containing a list of the JWS signing algorithms supported by the authorization server for the signature on the Client Attestation PoP JWT.
- * Change Controller: IETF
- * Reference: this specification

13.4. Registration of attest_jwt_client_auth Token Endpoint Authentication Method

This section registers the value "attest_jwt_client_auth" in the IANA "OAuth Token Endpoint Authentication Methods" registry established by OAuth 2.0 Dynamic Client Registration Protocol [RFC7591].

- * Token Endpoint Authentication Method Name: "attest_jwt_client_auth"
- * Change Controller: IESG
- * Specification Document(s): TBC

13.5. HTTP Field Name Registration

This section requests registration of the following scheme in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [IANA.HTTP.Fields] described in [RFC9110]:

- * Field Name: OAuth-Client-Attestation
- * Status: permanent
- * Structured Type: Item
- * Reference: Section 6.1 of this specification

- * Field Name: OAuth-Client-Attestation-PoP
- * Status: permanent
- * Structured Type: Item
- * Reference: Section 6.1 of this specification

- * Field Name: OAuth-Client-Attestation-Challenge
- * Status: permanent
- * Structured Type: Item
- * Reference: Section 7 of this specification

14. References

14.1. Normative References

[IANA.HTTP.Fields]

IANA, "Hypertext Transfer Protocol (HTTP) Field Name Registry", n.d., <<https://www.iana.org/assignments/http-fields/http-fields.xhtml>>.

[IANA.JOSE.ALGS]

IANA, "JSON Web Signature and Encryption Algorithms", n.d., <<https://www.iana.org/assignments/jose/jose.xhtml#web-signature-encryption-algorithms>>.

[IANA.OAuth.Params]

IANA, "OAuth Authorization Server Metadata", n.d.,
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#authorization-server-metadata>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [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/rfc/rfc6750>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/rfc/rfc7519>>.
- [RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <<https://www.rfc-editor.org/rfc/rfc7591>>.
- [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/rfc/rfc7800>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <<https://www.rfc-editor.org/rfc/rfc8414>>.
- [RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <<https://www.rfc-editor.org/rfc/rfc8725>>.

- [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/rfc/rfc9110>>.
- [RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <<https://www.rfc-editor.org/rfc/rfc9112>>.
- [RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <<https://www.rfc-editor.org/rfc/rfc9126>>.

14.2. Informative References

- [ARF] "The European Digital Identity Wallet Architecture and Reference Framework", n.d..
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/rfc/rfc6749>>.
- [RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <<https://www.rfc-editor.org/rfc/rfc7523>>.
- [RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation procedures (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <<https://www.rfc-editor.org/rfc/rfc9334>>.
- [RFC9901] Fett, D., Yasuda, K., and B. Campbell, "Selective Disclosure for JSON Web Tokens", RFC 9901, DOI 10.17487/RFC9901, November 2025, <<https://www.rfc-editor.org/rfc/rfc9901>>.

Appendix A. Document History

-08

- * remove concatenated Serialization for Client Attestations
- * update all examples (removal of iss and nbf)
- * remove iss from Client Attestation JWT and Client Attestation PoP JWT

- * add small security consideration sub-section for MAC-based deployments
- * remove public clients reference and clarify this draft targets confidential clients
- * clarify this may be a client authentication mechanism but also may be not
- * add examples for RS usage and non client authentication
- * Add note on protocols providing a challenge on previous responses
- * add structured-type to iana header field registration requests
- * moving Authorization Server metadata into it's own top level section
- * editorial fixes

-07

- * remove restrictions to not allow MAC-based algorithms
- * require iat in Client Attestation PoP JWT
- * clarify use_attestation_challenge and add invalid_client_attestation
- * add client_attestation_signing_alg_values_supported and client_attestation_pop_signing_alg_values_supported to IANA registration
- * add implementation consideration for Authorization Server Metadata
- * clarify refresh token binding
- * check client_id at PAR endpoint
- * added use_fresh_attestation as an error to signal that the attestation was not deemed fresh enough by the server
- * mandate the defined header fields if the attestation and pop are transferred via header fields

-06

- * clarify client_id processing in token request with client attestation
- * clarify usage of client attestation outside of oauth2 applications
- * add oauth error response values invalid_client_attestation and use_attestation_challenge
- * revert the HTTP OPTIONS mechanism to fetch nonces and add a dedicated challenge endpoint
- * rename nonce to challenge
- * rewrite security consideration on replay attacks
- * add implementation consideration on replay attacks
- * remove exp from Client Attestation PoP JWT
- * add verification and processing rules

-05

- * add nonce endpoint
- * add metadata entry for nonce
- * improve introduction
- * rename client backend to client attester
- * fix missing typ header in examples

-04

- * remove key attestation example
- * restructured JWT Claims for better readability
- * added JOSE typ values for Client Attestation and Client Attestation PoP
- * add RATS relation
- * add concatenated representation without headers
- * add PAR endpoint example

- * fix PoP examples to include jti and nonce

- * add iana http field name registration

-03

- * remove usage of RFC7521 and the usage of client_assertion

- * add new header-based syntax introducing Oauth-Client-Attestation and OAuth-Client-Attestation-PoP

- * add Client Instance to the terminology and improve text around this concept

-02

- * add text on the inability to rotate the Client Instance Key

-01

- * Updated eIDAS example in appendix

- * Removed text around jti claim in client attestation, refined text for its usage in the client attestation pop

- * Refined text around cnf claim in client attestation

- * Clarified how to bind refresh tokens to a Client Instance using this client authentication method

- * Made it more explicit that the client authentication mechanism is general purpose making it compatible with extensions like PAR

- * Updated acknowledgments

- * Simplified the diagram in the introduction

- * Updated references

- * Added some guidance around replay attack detection

-00

- * Initial draft

Acknowledgments

We would like to thank Babis Routis, Brian Campbell, Dimitris Zarras, Filip Skokan, Francesco Marino, Frederik Krogsdal Jacobsen, Guiseppe De Marco, Joseph Heenan, Kristina Yasuda, Micha Kraus, Michael B. Jones, Takahiko Kawasaki, Timo Glastra and Torsten Lodderstedt for their valuable contributions to this specification.

Authors' Addresses

Tobias Looker
MATTR
Email: tobias.looker@mattr.global

Paul Bastian
Bundesdruckerei
Email: paul.bastian@bdr.de

Christian Bormann
SPRIND
Email: chris.bormann@gmx.de