Web Authorization Protocol
Internet-Draft
Intended status: Informational
Expires: 8 January 2026

T. Looker
MATTR
P. Bastian
Bundesdruckerei
C. Bormann
SPRIND

7 July 2025

OAuth 2.0 Attestation-Based Client Authentication
draft-ietf-oauth-attestation-based-client-auth-06

Abstract

This specification defines an extension to the OAuth 2 protocol as defined in [RFC6749] which enables a Client Instance to include a key-bound attestation in interactions with an Authorization Server or a Resource Server. This new method enables Client Instances involved in a client deployment that is traditionally viewed as a public client, to be able to utilize this key-bound attestation to authenticate.

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.

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

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 10
   6.1. Client Attestation HTTP Headers 10
   6.2. Validating HTTP requests feature client attestations . . 11
   6.3. Client Attestation at the Token Endpoint 12
   6.4. Client Attestation at the PAR Endpoint 13
7. Concatenated Serialization for Client Attestations 13
   7.1. Concatenated Serialization Format 14
   7.2. Validating the Concatenated Serialization 14
8. Challenge Retrieval 15
   8.1. Providing Challenges on Previous Responses 16
9. Verification and Processing 16
10. Implementation Considerations 17
   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
13. Appendix A IANA Considerations 21
13.1. OAuth Parameters Registration 21
13.2. OAuth Extensions Error Registration 21
13.3. Registration of attest_jwt_client_auth Token Endpoint Authentication Method 22
13.4. HTTP Field Name Registration 22
14. References 23
14.1. Normative References 23
14.2. Informative References 24
Appendix A. Document History 25
Acknowledgments 27
Authors' Addresses 27

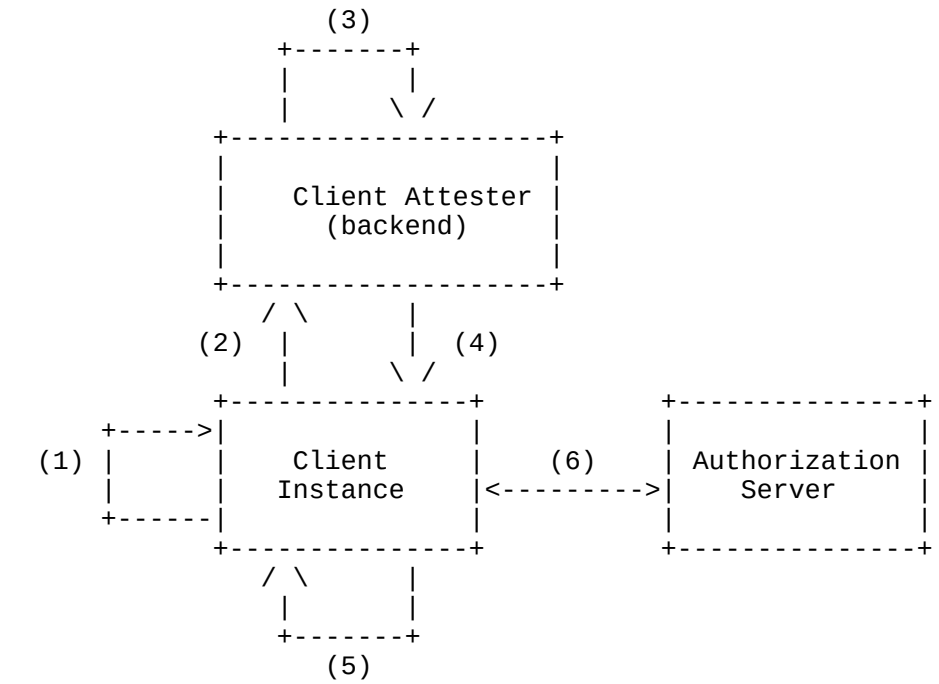
1. Introduction

Traditional OAuth security concepts perform client authentication through a backend channel. In ecosystems such as the Issuer-Holder-Verifier model used in [SD-JWT], 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 public clients to authenticate securely and reliably has become increasingly important. Leveraging platform mechanisms to validate a client instance, e.g. for 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 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 can be 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, or via a concatenated serialization (as described in Section 7) to enable usage outside of OAuth2 based interactions.

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:

* iss: REQUIRED. The iss (subject) claim MUST contains a unique identifier for the entity that issued the JWT. In the absence of an application profile specifying otherwise, compliant applications MUST compare issuer values using the Simple String Comparison method defined in Section 6.2.1 of [RFC3986].

* 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.

* nbf: OPTIONAL. The nbf (not before) claim MUST specify the time before which the Client Attestation MUST NOT be accepted for processing.

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 the JWT if it is using a Message Authentication Code (MAC) based algorithm. The authorization server MUST reject JWTs with an invalid signature.

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"
}
.
{
  "iss": "https://attester.example.com",
  "sub": "https://client.example.com",
  "nbf": 1300815780,
  "exp": 1300819380,
  "cnf": {
    "jwk": {
      "kty": "EC",
      "use": "sig",
      "crv": "P-256",
      "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
      "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
    }
  }
}
```

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:

* iss: REQUIRED. The iss (issuer) claim MUST specify client_id value of the OAuth Client.

* 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.

* 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.

* iat: OPTIONAL. 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.

* nbf: OPTIONAL. The nbf (not before) claim MUST specify the time before which the Client Attestation PoP MUST NOT be accepted for processing.

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 the JWT if it is using a Message Authentication Code (MAC) based 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 value of the iss claim, representing the client_id MUST match the value of the sub claim in the corresponding Client Attestation JWT.

5. 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"
}
.
{
  "iss": "https://client.example.com",
  "aud": "https://as.example.com",
  "nbf":1300815780,
  "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

A Client Attestation JWT and Client Attestation PoP JWT can be included in an HTTP request using the following request header fields.

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: eyJ0eXAiOiJvYXV0aC1jbGllbnQtYXR0ZXN0YXRpb24
rand0IiwiYWxnIjoiRVMyNTYiLCJraWQiOiIxMSJ9.eyJpc3MiOiJodHRwczovL2F0dGV
zdGVyLmV4YW1wbGUuY29tIiwic3ViIjoiaHR0cHM6Ly9jbGllbnQuZXhhbXBsZS5jb20i
LCJuYmYiOjEzMDA4MTU3ODAsImV4cCI6MTMwMDgxOTM4MCwiY25mIjp7Imp3ayI6eyJrd
HkiOiJFQyIsInVzZSI6InNpZyIsImNydiI6IlAtMjU2IiwieCI6IjE4d0hMZUlnVzl3Vk
42VkQxVHhncHF5MkxzellrTWY2SjhualZBaWJ2aE0iLCJ5IjoiLVY0ZFM0VWFMTWdQXzR
mWTRqOGlyN2NsMVRYbEZkQWdjeDU1bzdUa2NTQSJ9fX0.4bCswkgmUHw06kKdiS2KEySR
gjj73yCEIcrz3Mv7Bgns4Bm1tCQ9FAqMLtgzb5NthwJT9AhAEBogbiD5DtxV1g
```

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

```
OAuth-Client-Attestation-PoP: eyJhbGciOiJFUzI1NiIsInR5cCI6Im9hdXRoLWN
saWVudC1hdHRlc3RhdGlvbi1wb3Arand0In0.eyJpc3MiOiJodHRwczovL2NsaWVudC5l
eGFtcGxlLmNvbSIsImF1ZCI6Imh0dHBzOi8vYXMuZXhhbXBsZS5jb20iLCJuYmYiOjEzM
DA4MTU3ODAsImV4cCI6MTMwMDgxOTM4MCwianRpIjoiZDI1ZDAwYWItNTUyYi00NmZjLW
FlMTktOThmNDQwZjI1MDY0Iiwibm9uY2UiOiI1YzFhOWUxMC0yOWZmLTRjMmItYWU3My0
1N2MwOTU3YzA5YzQifQ.rEa-dKJgRuD-aI-4bj4fDGH1up4jV--IgDMFdb9A5jSSWB7Uh
HfvLOVU_ZvAJfOWfO0MXyeunwzM3jGLB_TUkQ
```

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 feature 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.

An error parameter according to Section 3 of [RFC6750] SHOULD be included to indicate why a request was declined. If the Client Attestation is absent or not using an expected server-provided challenge, the value use_attestation_challenge can be used to

indicate that an attestation with a server-provided challenge was expected. If the attestation and proof of possession was present but could not be successfully verified, the value invalid_client_attestation is used.

6.3. Client Attestation at the Token Endpoint

While usage of the the client attestation mechanism defined by this draft can be used in a variety of different HTTP requests to different endpoints, usage within the token request as defined by [RFC6749] has particular additional considerations outlined below.

The Authorization Server MUST perform all of the checks outlined in Section 6.2 for a received access token request which is making use of the client attestation mechanism as defined by this draft.

If the token request contains a client_id parameter as per [RFC6749] the Authorization Server MUST verify that the value of this parameter is the same as the client_id value in the sub claim of the Client Attestation and iss claim of the Client Attestation PoP.

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

```
POST /token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
OAuth-Client-Attestation: eyJ0eXAiOiJvYXV0aC1jbGllbnQtYXR0ZXN0YXRpb24
rand0IiwiYWxnIjoiRVMyNTYiLCJraWQiOiIxMSJ9.eyJpc3MiOiJodHRwczovL2F0dGV
zdGVyLmV4YW1wbGUuY29tIiwic3ViIjoiaHR0cHM6Ly9jbGllbnQuZXhhbXBsZS5jb20i
LCJuYmYiOjEzMDA4MTU3ODAsImV4cCI6MTMwMDgxOTM4MCwiY25mIjp7Imp3ayI6eyJrd
HkiOiJFQyIsInVzZSI6InNpZyIsImNydiI6IlAtMjU2IiwieCI6IjE4d0hMZUlnVzl3Vk
42VkQxVHhncHF5MkxzellrTWY2SjhualZBaWJ2aE0iLCJ5IjoiLVY0ZFM0VWFMTWdQXzR
mWTRqOGlyN2NsMVRYbEZkQWdjeDU1bzdUa2NTQSJ9fX0.4bCswkgmUHw06kKdiS2KEySR
gjj73yCEIcrz3Mv7Bgns4Bm1tCQ9FAqMLtgzb5NthwJT9AhAEBogbiD5DtxV1g
OAuth-Client-Attestation-PoP: eyJhbGciOiJFUzI1NiIsInR5cCI6Im9hdXRoLWN
saWVudC1hdHRlc3RhdGlvbi1wb3Arand0In0.eyJpc3MiOiJodHRwczovL2NsaWVudC5l
eGFtcGxlLmNvbSIsImF1ZCI6Imh0dHBzOi8vYXMuZXhhbXBsZS5jb20iLCJuYmYiOjEzM
DA4MTU3ODAsImV4cCI6MTMwMDgxOTM4MCwianRpIjoiZDI1ZDAwYWItNTUyYi00NmZjLW
FlMTktOThmNDQwZjI1MDY0IiwibmJ9uY2UiOiI1YzFhOWUxMC0yOWZmLTRjMmItYWU3My0
01N2MwOTU3YzA5YzQifQ.rEa-dKJgRuD-aI-4bj4fDGH1up4jV--IgDMFdb9A5jSSWB7Uh
HfvLOVU_ZvAJfOWfO0MXyeunwzM3jGLB_TUkQ

grant_type=authorization_code&
code=n0esc3NRze7LTCu7iYzS6a5acc3f0ogp4
```

6.4. Client Attestation at the PAR Endpoint

A Client Attestation can be used at the PAR endpoint instead of alternative client authentication mechanisms like JWT client assertion-based authentication (as defined in Section 2.2 of [RFC7523]).

The Authorization Server MUST perform all of the checks outlined in Section 6.2 for a received PAR request which is making use of the client attestation mechanism as defined by this draft.

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

```
POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
OAuth-Client-Attestation: eyJ0eXAiOiJvYXV0aC1jbGllbnQtYXR0ZXN0YXRpb24
rand0IiwiYWxnIjoiRVMyNTYiLCJraWQiOiIxMSJ9.eyJpc3MiOiJodHRwczovL2F0dGV
zdGVyLmV4YW1wbGUuY29tIiwic3ViIjoiaHR0cHM6Ly9jbGllbnQuZXhhbXBsZS5jb20i
LCJuYmYiOjEzMDA4MTU3ODAsImV4cCI6MTMwMDgxOTM4MCwiY25mIjp7Imp3ayI6eyJrd
HkiOiJFQyIsInVzZSI6InNpZyIsImNydiI6IlAtMjU2IiwieCI6IjE4d0hMZUlnVzl3Vk
42VkQxVHhncHF5MkxzellrTWY2SjhualZBaWJ2aE0iLCJ5IjoiLVY0ZFM0VWFMTWdQXzR
mWTRqOGlyN2NsMVRYbEZkQWdjeDU1bzdUa2NTQSJ9fX0.4bCswkgmUHw06kKdiS2KEySR
gjj73yCEIcrz3Mv7Bgns4Bm1tCQ9FAqMLtgzb5NthwJT9AhAEBogbiD5DtxV1g
OAuth-Client-Attestation-PoP: eyJhbGciOiJFUzI1NiIsInR5cCI6Im9hdXRoLWN
saWVudC1hdHRlc3RhdGlvbi1wb3Arand0In0.eyJpc3MiOiJodHRwczovL2NsaWVudC5l
eGFtcGxlLmNvbSIsImF1ZCI6Imh0dHBzOi8vYXMuZXhhbXBsZS5jb20iLCJuYmYiOjEzM
DA4MTU3ODAsImV4cCI6MTMwMDgxOTM4MCwianRpIjoiZDI1ZDAwYWItNTUyYi00NmZjLW
FlMTktOThmNDQwZjI1MDY0Iiwibm9uY2UiOiI1YzFhOWUxMC0yOWZmLTRjMmItYWU3My0
1N2MwOTU3YzA5YzQifQ.rEa-dKJgRuD-aI-4bj4fDGH1up4jV--IgDMFdb9A5jSSWB7Uh
HfvLOVU_ZvAJfOWfO0MXyeunwzM3jGLB_TUkQ

response_type=code&state=af0ifjsldkj&client_id=s6BhdRkqt3
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
&code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoeK1t8U
&code_challenge_method=S256&scope=account-information
```

7. Concatenated Serialization for Client Attestations

A Client Attestation according to this specification MAY be presented using an alternative representation for cases where the header-based mechanism (as introduced in introduced in Section 6.1 does not fit the underlying protocols, e.g., for direct calls to Browser APIs. In those cases, a concatenated serialization of the Client Attestation and Client Attestation PoP can can be used.

7.1. Concatenated Serialization Format

This representation is created by concatenating Client Attestation and Client Attestation PoP separated by a tilde (’~’) character:

<Client Attestation>~<Client Attestation PoP>

This form is similar to an SD-JWT+KB according to Section 5 of [SD-JWT] but does not include Disclosures, uses different typ values and does not include the sd_hash claim in the PoP.

This concatenated serialization form allows a the presentation of a Client Attestation and Client Attestation PoP for cases where a header-based approach is unavailable, e.g., to establish trust in a client when using a direct Browser API call.

The following is an example of such a concatenated serialization (with extra line breaks for display purposes only):

eyJ0eXAiOiJvYXV0aC1jbGllbnQtYXR0ZXN0YXRpb24rand0IiwiYWxnIjoiRVMyNTYiL
CJraWQiOiIxMSJ9.eyJpc3MiOiJodHRwczovL2F0dGVzdGVyLmV4YW1wbGUuY29tIiwic
3ViIjoiaHR0cHM6Ly9jbGllbnQuZXhhbXBsZS5jb20iLCJuYmYiOjEzMDA4MTU3ODAsIm
V4cCI6MTMwMDgxOTM4MCwiY25mIjp7Imp3ayI6eyJrdHkiOiJFQyIsInVzZSI6InNpZyI
sImNydiI6IlAtMjU2IiwieCI6IjE4d0hMZUlnVzl3Vk42VkQxVHhncHF5MkxzellrTWY2
SjhualZBaWJ2aE0iLCJ5IjoiLVY0ZFM0VWFMTWdQXzRmWTRqOGlyN2NsMVRYbEZkQWdje
DU1bzdUa2NTQSJ9fX0.4bCswkgmUHw06kKdiS2KEySRgjj73yCEIcrz3Mv7Bgns4Bm1tC
Q9FAqMLtgzb5NthwJT9AhAEBogbiD5DtxV1g~eyJhbGciOiJFUzI1NiIsInR5cCI6Im9h
dXRoLWNsaWVudC1hdHRlc3RhdGlvbi1wb3Arand0In0.eyJpc3MiOiJodHRwczovL2Nsa
WVudC5leGFtcGxlLmNvbSIsImF1ZCI6Imh0dHBzOi8vYXMuZXhhbXBsZS5jb20iLCJuYm
YiOjEzMDA4MTU3ODAsImV4cCI6MTMwMDgxOTM4MCwianRpIjoiZDI1ZDAwYWItNTUyYi0
0NmZjLWFlMTktOThmNDQwZjI1MDY0Iiwibm9uY2UiOiI1YzFhOWUxMC0yOWZmLTRjMmIt
YWU3My01N2MwOTU3YzA5YzQifQ.rEa-dKJgRuD-aI-4bj4fDGH1up4jV--IgDMFdb9A5j
SSWB7UhHfvLOVU_ZvAJfOWfO0MXyeunwzM3jGLB_TUkQ

7.2. Validating the Concatenated Serialization

To validate a client attestation using the concatenated serialization form, the receiving server MUST ensure the following:

1. Before the ’~’ character, there exists precisely a single well-formed JWT conforming to the syntax outlined in Section 5.1.

2. After the ’~’ character, there exists precisely a single well-formed JWT conforming to the syntax outlined in Section 5.2.

3. The signature of the Client Attestation PoP JWT obtained after the '~' character verifies with the Client Instance Key contained in the cnf claim of the Client Attestation JWT obtained before the '~' character.

8. 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 terminology. 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 (#client-attestation-pop-jwt).

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 (#client-attestation-pop-jwt). 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

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

8.1. 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": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

9. 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 not MAC based, is supported by the application, and is acceptable per local policy.

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 and the iss claim of the Client Attestation PoP JWT.

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 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 form 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 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 chose 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.

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.

13. Appendix A 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 8 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: 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: this specification

13.3. 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

13.4. 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

* Reference: Section 6.1 of this specification

* Field Name: OAuth-Client-Attestation-PoP

* Status: permanent

* Reference: Section 6.1 of this specification

* Field Name: OAuth-Client-Attestation-Challenge

* Status: permanent

* Reference: Section 8 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>.

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>.

[SD-JWT] Fett, D., Yasuda, K., and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)", Work in Progress, Internet-Draft, draft-ietf-oauth-selective-disclosure-jwt-22, 29 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-selective-disclosure-jwt-22>.

Appendix A. Document History

-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 Brian Campbell, Filip Skokan, Francesco Marino, Guiseppe De Marco, Kristina Yasuda, Michael B. Jones, Takahiko Kawasaki 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@posteo.de

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