Web Authorization Protocol
Internet-Draft
Intended status: Informational
Expires: 17 April 2026

Y. Rosomakho
L. Weith
Zscaler

14 October 2025

Separating DPoP Bindings for Access and Refresh Tokens
draft-rosomakho-oauth-dpop-rt-00

Abstract

This document defines an extension to OAuth 2.0 Demonstrating Proof-of-Possession at the application level (DPoP, RFC 9449) that allows refresh tokens to be bound to a different proof-of-possession key than access tokens. In the existing specification, a single DPoP proof is used to bind both tokens to the same key material. However, in many deployments, refresh tokens and access tokens are stored and managed in different security contexts. To support this operational separation, this document introduces a new HTTP header field, DPoP-RT, and corresponding DPoP-RT-Nonce mechanism, enabling independent proof-of-possession for refresh token use while preserving compatibility with existing DPoP-bound access tokens.

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://yaroslavros.github.io/oauth-dpop-rt/draft-rosomakho-oauth-dpop-rt.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rosomakho-oauth-dpop-rt/.

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/yaroslavros/oauth-dpop-rt.

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 17 April 2026.

Copyright Notice

Table of Contents

1. Introduction

OAuth 2.0 Demonstrating Proof-of-Possession [DPoP] defines an application-level mechanism by which a client proves possession of a cryptographic key when obtaining and using access tokens. The same proof is applied when the client requests new tokens from the authorization server (AS), binding both access and refresh tokens to the same DPoP key material.

In practice, many deployments store and manage refresh tokens and access tokens in different security domains. For example, refresh tokens are often retained by a secure backend service or hardware-protected component, while access tokens are consumed by short-lived front-end instances or mobile applications. Binding both token types to the same DPoP key limits operational flexibility, complicates key rotation, and increases the impact of key compromise.

This document defines a lightweight extension to DPoP that enables the refresh token to be bound to a distinct proof-of-possession key from the one used for access tokens. The mechanism introduces a new HTTP header field, DPoP-RT, which carries a signed JWT proving possession of the refresh token key when the client redeems a refresh token at the token endpoint. An optional DPoP-RT-Nonce response header provides replay protection and freshness guarantees analogous to the existing DPoP-Nonce mechanism.

This separation allows deployments to:

* Isolate long-term refresh token credentials from short-lived access token keys

* Rotate or revoke access token keys without affecting the refresh token flow

* Reduce the blast radius of access token key compromise

The extension is fully backward compatible with [DPoP] as clients and authorization servers that do not implement this specification continue to operate unchanged, while those that support DPoP-RT can negotiate its use on a per-client basis.

A motivating scenario for this extension is an agent managing numerous long-lived tokens on behalf of its users, where user participation is often required for revocation. When combined with a hardware security module (HSM), DPoP allows use of a refresh token to be halted immediately by disabling the associated key material. The token remains valid but unusable until the key is reactivated. In large, distributed systems with many worker nodes, however, involving an HSM in every transaction is operationally impractical, motivating this extension.

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.

The terminology and conventions defined in [DPoP] apply. In addition, this document uses the following terms:

DPoP-RT proof: A JWT carried in the DPoP-RT HTTP header that proves possession of the refresh token key when a refresh token is presented at the token endpoint.

Refresh token key: The asymmetric key used to generate DPoP-RT proofs and to which a refresh token is bound.

Access token key: The asymmetric key used to generate DPoP proofs and to which access tokens are bound (as defined in [DPoP]).

DPoP-RT-Nonce: An authorization-server-generated value used to ensure freshness of DPoP-RT proofs, analogous to the DPoP-Nonce defined in [DPoP].

3. Overview

This specification extends [DPoP] by introducing a second proof-of-possession header, DPoP-RT, which is used to bind refresh tokens to a distinct key from the one used for access tokens. The mechanism allows authorization servers (AS) and clients to maintain separate key lifecycles for access and refresh tokens while preserving backward compatibility with existing DPoP deployments.

When a client initially requests tokens (for example, during an authorization code exchange), it MAY include both:

* a DPoP header — proving possession of the access token key, and

* a DPoP-RT header — proving possession of the refresh token key to which any issued refresh token will be bound.

If the DPoP-RT header is omitted, the AS follows the behavior defined in [DPoP] and binds both tokens to the same key from the DPoP proof.

When the client later uses the refresh token to obtain a new access token, it includes a new DPoP-RT proof (signed by refresh key) and a DPoP proof (signed by access key) if a DPoP-bound access token is to be issued. The authorization server MUST NOT infer the access token key from earier tokens.

The AS validates the DPoP-RT proof before processing the refresh request. If validation succeeds, the AS issues a new access token bound to the key demonstrated in the DPoP header, and, optionally, a new refresh token bound to the key demonstrated in the DPoP-RT header.

To ensure freshness and replay protection, the AS may provide a DPoP-RT-Nonce response header, whose value the client includes as a nonce claim in subsequent DPoP-RT proofs. This nonce mechanism operates independently from the DPoP-Nonce defined in [DPoP].

The following diagram illustrates a complete flow:

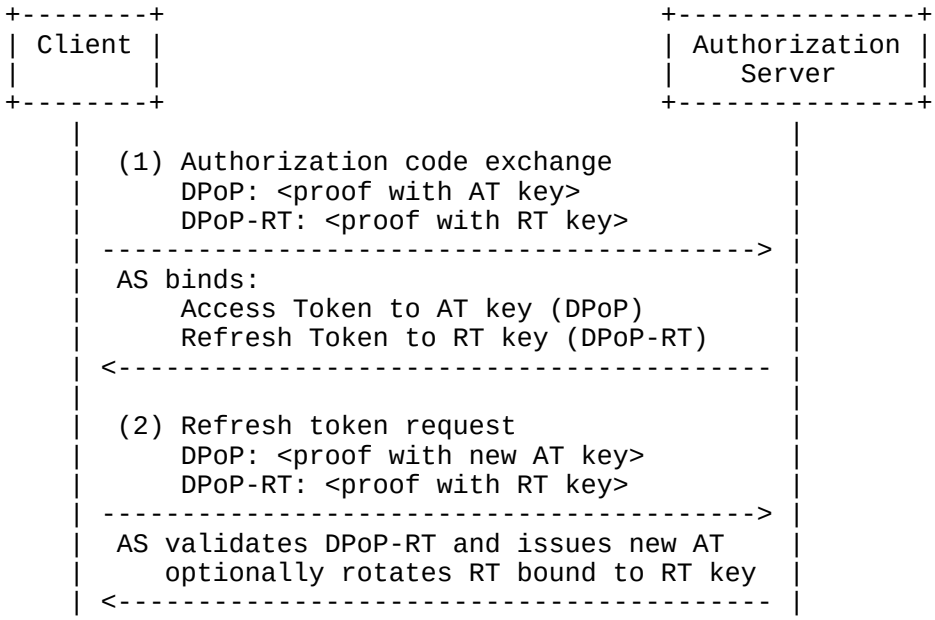

Figure 1: Overall DPoP and DPoP-RT flow

This explicit dual-key model ensures that each token binding key is known and verifiable by the AS at issuance time, simplifying validation and preventing rebinding attacks during refresh operations.

4. Client Use of DPoP-RT

This section defines how clients use the new DPoP-RT HTTP header field and its relationship to the existing DPoP header when interacting with the authorization server (AS).

4.1. Including DPoP-RT during Token Issuance

When a client requests tokens from the AS (for example, during an authorization code exchange) with DPoP mechanism in use, it:

* MUST include a DPoP header containing a proof signed with the access token key; and

* MAY include a DPoP-RT header containing a proof signed with the refresh token key.

If a valid DPoP-RT proof is provided, the AS MUST bind any issued refresh token to the key represented by that proof. If DPoP-RT is omitted, the AS MUST bind both the access token and refresh token to the key conveyed in the DPoP proof, following the behavior defined in [DPoP].

4.2. Including DPoP-RT during Token Refresh

When using the refresh token to obtain a new access token, the client MUST include both:

1. A DPoP-RT header proving possession of the refresh token key, and

2. A DPoP header proving possession of the access token key to which the new access token will be bound.

The AS MUST NOT infer the new access token key from any previously issued tokens or server-side state. The only valid source for the access token binding key in a refresh request is the key proved in the DPoP header of that request.

If the DPoP header is absent, the AS MUST NOT issue a DPoP-bound access token. It MAY issue an unbound (bearer) access token only if explicitly permitted by policy or by the client registration metadata (e.g., when dpop_bound_access_tokens is false). Otherwise, the request MUST be rejected according to the rules in [DPoP].

If the DPoP-RT header is absent or invalid and the client is registered with dpop_bound_refresh_tokens set to true, the AS MUST reject the request using the invalid_dpop_rt_proof error.

4.3. Example Requests

Authorization code exchange with dual proofs:

```
POST /oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
DPoP: eyJhbGciOiJFUzI1NiIsInR5cCI6ImRwb3Araj... (AT key)
DPoP-RT: eyJhbGciOiJFUzI1NiIsInR5cCI6ImRwb3AtcnQran... (RT key)

grant_type=authorization_code&
code=SplxlOBeZQQYbYS6WxSbIA&
redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

Figure 2: Authorization code exchange with dual proofs

Refresh token exchange:

```
POST /oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
DPoP: eyJhbGciOiJFUzI1NiIsInR5cCI6ImRwb3Araj... (new AT key)
DPoP-RT: eyJhbGciOiJFUzI1NiIsInR5cCI6ImRwb3AtcnQran... (RT key)

grant_type=refresh_token&
refresh_token=eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVC...
```

Figure 3: Refresh token exchange with dual proofs

4.4. Use of Nonces

The DPoP-RT proof may contain a nonce claim when the AS requires proof freshness. If the AS returns a DPoP-RT-Nonce response header, the client MUST include that value in the nonce claim of the next DPoP-RT proof it constructs.

The DPoP-RT-Nonce mechanism is independent from the DPoP-Nonce defined in [DPoP]; each proof type maintains its own nonce sequence and replay protection.

5. DPoP-RT Proof Structure

A DPoP-RT proof is a JSON Web Token [JWT] carried in the DPoP-RT HTTP header. It proves possession of the refresh token key when a client presents or obtains a refresh token at the authorization server (AS).

The proof follows the same overall structure and construction rules as the DPoP proof defined in [DPoP], but with several distinct requirements.

5.1. JWS Header

The JOSE Header of a DPoP-RT JWT MUST contain at least the following parameters:

typ: A field with the value dpop-rt+jwt

alg: An identifier for a JWS asymmetric digital signature algorithm. It MUST NOT be none or an identifier for a symmetric algorithm. It MAY differ from algorithm used in DPoP JWT.

jwk: Represents the public key chosen by the client in JSON Web Key [JWK] format as defined in Section 4.1.3 of [JWK]. It MUST NOT contain a private key.

5.2. JWT Claims

The payload of a DPoP-RT proof MUST contain at least the following claims:

jti: Unique identifier for the DPoP-RT proof JWT. It MUST NOT match any other DPoP or DPoP-RT proof used in the same context during the time window of validity.

htm: The value of the HTTP method (Section 9.1 of [HTTP]) of the request to which the JWT is attached.

htu: The HTTP target URI (Section 7.1 of [HTTP]) of the request to which the JWT is attached, without query and fragment parts.

iat: Creation timestamp of the JWT (Section 4.1.6 of [JWT]).

When refresh token is presented alongside with DPoP-RT proof, it MUST contain the following claim

rth: Hash of the refresh token. The value MUST be the result of a base64url encoding (as defined in Section 2 of [JWS]) the SHA-256 hash of the ASCII encoding of the associated refresh token value.

When the AS provides a DPoP-RT-Nonce HTTP header in a response, the DPoP proof MUST also contain the following claim:

nonce: A recent nonce provided via the DPoP-RT-Nonce HTTP header.

A DPoP-RT proof MAY contain other JOSE Header Parameters or claims as defined by extension, profile, or deployment-specific requirements.

5.3. Construction and Signing

To provide a DPoP-RT proof a client:

1. Creates a JWT with the header and claims described above.

2. Computes the base64url-encoded SHA-256 digest of the refresh-token value and include it in the rth claim.

3. Signs the JWT with the private key corresponding to the public key in the jwk header parameter.

4. Includes the resulting compact serialization value in the DPoP-RT HTTP header.

A conceptual example of decoded content of DPoP-RT proof:

```
{
  "typ":"dpop-rt+jwt",
  "alg":"ES256",
  "jwk": {
    "kty":"EC",
    "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv":"P-256"
  }
}
.
{
  "jti":"f62b7f84-eafd-44d8-88b3-ba4f6a574744",
  "htm":"POST",
  "htu":"https://as.example.com/oauth2/token",
  "iat":1760400097,
  "rth":"fUHyO2r2Z3DZ53EsNrWBb0xWXoaNy59IiKCAqksmQEo"
}
```


Figure 4: Example JWT Content of a DPoP-RT Proof

6. Authorization Server Processing

This section defines the processing requirements for authorization servers (AS) when handling requests that include the DPoP-RT header.

6.1. Proof Validation

When a request containing a DPoP-RT header is received, the AS MUST perform the following checks in order:

1. Verify the JWS signature using the public key contained in the jwk header parameter. The algorithm MUST match the alg header value, and the algorithm MUST be supported and permitted by the AS.

2. Confirm that the typ header value is "dpop-rt+jwt". Proofs with any other value MUST be rejected.

3. Ensure that the htm and htu claims exactly match the HTTP method and absolute URI of the current request to the token endpoint. Implementations MUST compare the scheme, host, and path components in a case-sensitive manner as defined in [DPoP].

4. Validate that the iat claim is within an acceptable time window and that the jti value has not been used before with the same key and nonce context. The AS MUST maintain state or use another mechanism to prevent re-use of jti values for the lifetime of the nonce or replay window.

5. If the AS previously issued a DPoP-RT-Nonce value, it MUST verify that the proof contains a matching nonce claim. If the nonce is missing, invalid, or expired, the AS MUST reject the request with the use_dpop_rt_nonce error and include a new DPoP-RT-Nonce response header.

6. If the request includes a refresh_token parameter, the AS MUST compute the base64url-encoded SHA-256 digest of the presented refresh token value and compare it to the rth claim in the proof. A mismatch or missing rth claim in this case MUST cause the request to fail with invalid_dpop_rt_proof. If no refresh token is included in the request (for example, during the initial authorization code exchange), the rth claim MUST NOT be present.

If any of the above checks fail, the AS MUST reject the request with an appropriate error.

Successful validation proves that the client possesses the private key corresponding to the refresh-token binding and that the proof is specific to the current token request.

6.2. Token Binding and Issuance

When both DPoP and DPoP-RT proofs are successfully validated, the AS MUST bind tokens as follows:

* The access token is bound to the public key carried in the jwk header of the DPoP proof, as specified in [DPoP].

* The refresh token is bound to the public key carried in the jwk header of the DPoP-RT proof.

If the client omits the DPoP-RT header, the AS MUST bind both tokens to the same key proved in the DPoP header, preserving compatibility with [DPoP]. If the client omits the DPoP header and policy allows bearer tokens, the AS MAY issue an unbound (bearer) access token. Otherwise, the request MUST be rejected according to [DPoP].

If the AS rotates refresh tokens as part of the response, the newly issued refresh token MUST be bound to the same refresh token key proven in the current request.

6.3. DPoP-RT Nonce Issuance

The AS MAY return a DPoP-RT-Nonce header in success or error response to signal that the client must include this value in subsequent DPoP-RT proofs. The AS SHOULD periodically refresh the nonce to limit replay potential. Nonce values are opaque to the client and MUST be treated as short-lived.

The DPoP-RT-Nonce mechanism operates independently from the DPoP-Nonce defined in [DPoP]. Each proof type maintains its own nonce sequence and state.

7. Client Registration Metadata

This specification introduces a new boolean client registration metadata parameter named dpop_bound_refresh_tokens.

With true value of this parameter client indicates that its refresh tokens are always bound to a proof-of-possession key, as demonstrated by a DPoP-RT proof, and that the AS MUST enforce the use of such proofs whenever a refresh token is presented.

This parameter can be supplied during client registration [OAUTH-DYNAMIC]. Authorization servers that support this extension MAY advertise support for this metadata in their documentation or discovery interfaces, but no new AS metadata fields are defined by this specification.

7.1. Server Enforcement

If a client is registered with dpop_bound_refresh_tokens = true, the AS MUST:

1. Reject any request using a refresh token that does not contain a valid DPoP-RT header. The AS MUST return the invalid_dpop_rt_proof error when a proof is malformed, invalid, or fails validation. The AS MUST return use_dpop_rt_nonce when a fresh proof with a valid nonce is required.

2. Bind all refresh tokens issued to that client to the key demonstrated by the corresponding DPoP-RT proof at issuance time.

3. AS MUST reject unbound refresh token use including refresh token previously issued without binding information (for example, before registration or migration).

If a client is registered with dpop_bound_refresh_tokens = false or the parameter is omitted, the AS MAY treat DPoP-RT as optional and continue to follow the behavior defined in [DPoP].

8. Security Considerations

This extension introduces an additional proof type (DPoP-RT) and the ability to bind refresh tokens to a distinct proof-of-possession key. Implementations MUST apply all security considerations of [DPoP] in addition to those described below.

8.1. Key Separation and Compartmentalization

Using distinct keys for access and refresh tokens limits the impact of key compromise:

* If the access token key is compromised, an attacker can invoke protected APIs until the access token expires but cannot obtain new tokens.

* If the refresh token key is compromised, an attacker can redeem the refresh token and obtain new access tokens, binding them to an attacker-controlled access token key. Separation merely forces AS interaction.

Implementers SHOULD store the refresh-token key in an HSM or hardened server-side environment and MAY allow the access-token key to reside in a more transient context such as a front-end instance or mobile device. Key separation materially reduces the blast radius of access token key compromise but does not reduce the impact of a full refresh token key compromise.

8.2. Replay Protection and Nonces

DPoP-RT proofs are bearer artifacts and are therefore susceptible to replay if intercepted. To mitigate this risk:

* Clients MUST include unique jti values and use short-lived proofs.

* Authorization servers MUST detect and reject replayed jti values within the nonce or replay window.

* When the AS requires additional freshness, it SHOULD use the DPoP-RT-Nonce mechanism defined in this document. The AS MUST return use_dpop_rt_nonce to request a fresh proof with the supplied nonce.

* Nonces for DPoP and DPoP-RT are independent and MUST NOT be interchanged.

9. IANA Considerations

This document makes the following registrations.

9.1. HTTP Header Field Registrations

IANA is requested to register the following HTTP header fields, as specified by this document, in the "Hypertext Transfer Protocol (HTTP) Field Name Registry":

DPoP-RT:

* Field name: DPoP-RT

* Status: Provisional (Permanent if approved)

* Reference: This document

DPoP-RT-Nonce:

* Field name: DPoP-RT-Nonce

* Status: Provisional (Permanent if approved)

* Reference: This document

9.2. OAuth Extensions Error Registry

IANA is requested to register the following entries to the "OAuth Extensions Error Registry":

invalid_dpop_rt_proof:

* Name: invalid_dpop_rt_proof

* Usage Location: token error response

* Protocol Extension: This document

* Change Controller: IETF

* Reference: This document

use_dpop_rt_nonce:

* Name: use_dpop_rt_nonce

* Usage Location: token error response

* Protocol Extension: This document

* Change Controller: IETF

* Reference: This document

9.3. OAuth Dynamic Client Registration Metadata

IANA is requested to register the following value in the IANA "OAuth Dynamic Client Registration Metadata" registry:

* Client Metadata Name: dpop_bound_refresh_tokens

* Client Metadata Description: Boolean value specifying whether the client always uses DPoP-RT for refresh token requests

* Change Controller: IETF

* Reference: This document

9.4. Media Type Registration

IANA is requested to register the application/dpop-rt+jwt media type in the "Media Types" registry.

* Type name: application

* Subtype name: dpop-rt+jwt

* Required parameters: n/a

* Optional parameters: n/a

* Encoding considerations: binary. A DPoP-RT JWT is a JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period (’.’) characters.

* Security considerations: This document

* Interoperability considerations: n/a

* Published specification: This document

* Applications that use this media type: Applications using this specification for application-level proof of possession of refresh tokens

* Fragment identifier considerations: n/a

* Additional information:

  - File extension(s): n/a

  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Yaroslav Rosomakho, yrosomakho@zscaler.com

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Yaroslav Rosomakho, yrosomakho@zscaler.com

* Change controller: IETF

10. References

10.1. Normative References

[DPoP] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

[HTTP] 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>.

[JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

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

[JWT] 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>.

[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>.

[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>.

10.2. Informative References

[OAUTH-DYNAMIC]
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>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com

Loren Weith
Zscaler
Email: lweith@zscaler.com