OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: 20 March 2026

S. Nandakumar
C. Jennings
Cisco Systems

16 September 2025

Application-Agnostic Demonstration Proof of Possession (DPoP) Framework
draft-nandakumar-oauth-dpop-proof-00

Abstract

This document describes a generic framework for Demonstrating Proof of Possession (DPoP) that extends beyond HTTP-specific implementations. Building upon RFC 9449, this framework provides a protocol-agnostic approach for sender-constraining tokens through cryptographic proof of possession, enabling secure token binding across various application protocols and contexts.

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/oauth-dpop-generic/draft-generic-dpop-framework.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-nandakumar-oauth-dpop-proof/.

Discussion of this document takes place on the OAuth Working Group 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/oauth-dpop-generic.

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/.

Copyright Notice

Table of Contents

1. Introduction

RFC 9449 [RFC9449] defines a mechanism for sender-constraining OAuth 2.0 tokens via a proof-of-possession mechanism on the application level. DPoP as defined in RFC 9449 has two separable parts: the first part (covered in sections 4, 5, 6, and 8) describes how a client obtains a DPoP-bound token from an authorization server, while the second part (covered in sections 7 and 9) describes how the client proves control of the private key associated with a DPoP token when accessing protected resources.

This specification defines bindings for Media Over QUIC Transport (MOQT) and presents a generic framework that abstracts the core concepts of DPoP into a protocol-agnostic approach. While keeping the first part unchanged, this framework generalizes the second part to enable proof-of-possession token binding for various application contexts beyond HTTP while maintaining the security properties established in RFC 9449.

1.1. Conventions and Terminology

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 terms "access token", "refresh token", "authorization server", "resource server", "client", and "public client" are defined by "The OAuth 2.0 Authorization Framework" [RFC6749].

The terms "JSON Web Token (JWT)", "JOSE Header", and "JWS" are defined in [RFC7519] and [RFC7515].

Application protocol: The protocol which uses DPoP tokens as proof of authorization. This may be HTTP (as in RFC 9449) or any other application-layer protocol that requires proof-of-possession token binding.

2. Protocol Overview and Concept

The Generic DPoP Framework extends the DPoP mechanism to work across various application protocols beyond HTTP. The basic flow involves a client obtaining a DPoP-bound token from an authorization server, then using that token with an application protocol by providing proof of possession of the associated private key.

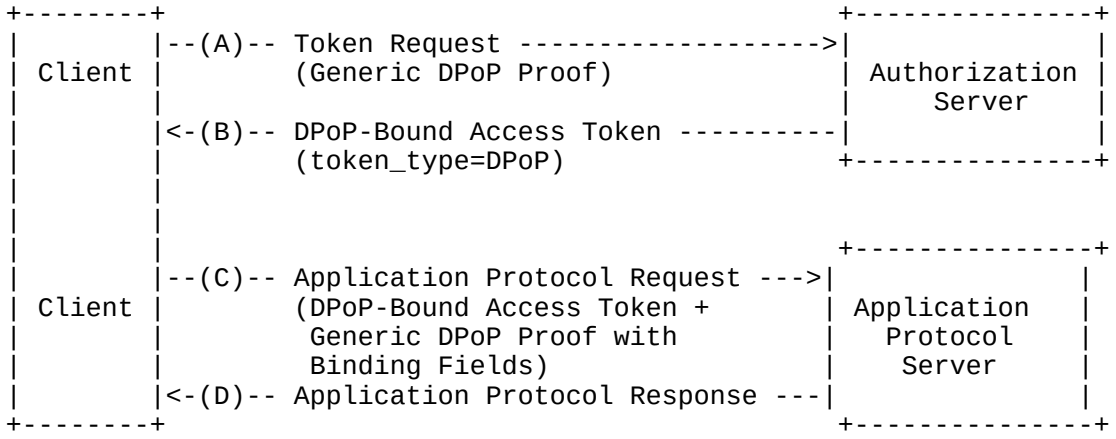

Figure 1: Generic DPoP Flow

The main data structure introduced by this specification is a generic DPoP proof JWT that can be used across various application protocols and contexts, as described in detail below. A client uses a generic DPoP proof JWT to prove the possession of a private key corresponding to a certain public key.

Roughly speaking, a generic DPoP proof is a signature over:

* protocol-specific authorization context data,

* a timestamp,

* a unique identifier,

* an optional server-provided nonce, and

* a hash of the associated access token when an access token is present within the request.

The application protocol needs to define binding fields that make the proof JWT unique to a specific protocol interaction. These binding fields replace the HTTP method (htm) and HTTP URI (htu) fields used in RFC 9449, providing protocol-specific context that prevents replay attacks across different protocol operations.

Key aspects of the protocol:

1. *Token Acquisition (Steps A-B)*: Uses the same mechanism as RFC 9449 sections 4, 5, 6, and 8

2. *Proof of Possession (Steps C-D)*: Generalizes RFC 9449 sections 7 and 9 with protocol-specific binding fields

3. *Binding Fields*: Each application protocol must define how to bind the proof to specific protocol operations to ensure uniqueness

The basic steps of a protocol flow with generic DPoP (without the optional nonce) are shown in Figure 1.

A. In the token request, the client sends an authorization grant (e.g., an authorization code, refresh token, etc.) to the authorization server in order to obtain an access token (and potentially a refresh token). The client attaches a generic DPoP proof to the request.

B. The authorization server binds (sender-constrains) the access token to the public key claimed by the client in the DPoP proof; that is, the access token cannot be used without proving possession of the respective private key. If a refresh token is issued to a public client, it is also bound to the public key of the DPoP proof.

C. To use the access token, the client has to prove possession of the private key by, again, providing a generic DPoP proof for that request to the protocol server. The protocol server needs to receive information about the public key to which the access token is bound. This information may be encoded directly into the access token (for JWT-structured access tokens) or provided via token introspection

endpoint (not shown). The protocol server verifies that the public key to which the access token is bound matches the public key of the DPoP proof. It also verifies that the access token hash in the DPoP proof matches the access token presented in the request.

D. The protocol server refuses to serve the request if the signature check fails or if the data in the DPoP proof is wrong, e.g., the authorization context does not match the expected context for the protocol operation. The access token itself, of course, must also be valid in all other respects.

The generic DPoP mechanism presented herein is not a client authentication method. In fact, a primary use case of DPoP is for public clients (e.g., single-page applications and applications on a user's device) that do not use client authentication. Nonetheless, generic DPoP is designed to be compatible with private_key_jwt and all other client authentication methods.

Generic DPoP does not directly ensure message integrity, but it relies on the underlying transport security layer (such as TLS) for that purpose. See Section 8 for details.

3. Application Protocol Requirements

Application protocols that wish to use this generic DPoP framework MUST define the following elements to ensure secure proof-of-possession token binding:

3.1. Binding Fields Definition

Each application protocol MUST specify:

1. *Required Binding Fields*: The mandatory fields within the Authorization Context (actx) object that uniquely identify and bind the proof to a specific protocol operation

2. *Field Semantics*: Clear definitions of what each binding field represents and how it relates to protocol messages and operations

3. *Uniqueness Requirements*: How the combination of binding fields ensures that each proof is unique to a specific protocol interaction

3.2. Nonce Support

Application protocols SHOULD provide a mechanism for servers to supply nonces to clients for replay protection, similar to Section 9 of RFC 9449. When nonce support is provided, the protocol specification MUST define:

1. How nonces are communicated from server to client
2. The lifetime and scope of nonces
3. How clients incorporate nonces into DPoP proofs

3.3. Security Requirements

Application protocol specifications MUST include:

1. *Replay Attack Prevention*: How the binding fields prevent replay of proofs across different protocol operations
2. *Cross-Protocol Security*: Measures to prevent proofs from being valid across different application protocols
3. *Protocol-Specific Threat Model*: Analysis of security threats specific to the application protocol context

3.4. Example Binding Implementation

For illustration, an application protocol might define binding fields such as:

* operation: The specific protocol operation being authorized
* resource_identifier: A unique identifier for the resource being accessed
* timestamp_context: Protocol-specific temporal context information

The combination of these fields would ensure that a DPoP proof is valid only for the specific operation, resource, and temporal context for which it was generated.

4. Application-Agnostic DPoP Proof Structure

This proof structure has the same general structure as a standard DPoP proof as defined in RFC 9449, with the key difference being that the HTTP-specific claims (htm for HTTP method and htu for HTTP URI) are replaced with the Authorization Context (actx) object to provide application protocol-specific binding information.

4.1. Authorization Context Object

The core extension introduced by this framework is the Authorization Context (actx) object, which replaces protocol-specific fields in the DPoP proof JWT payload.

A generic DPoP proof JWT MUST contain an Authorization Context object (actx) that specifies the protocol or context for which the proof is being generated.

4.1.1. Authorization Context Object Structure

The Authorization Context (actx) object MUST contain:

* type (string): A registered identifier specifying the protocol or context type

* Additional fields as defined by the specific context type specification

```
{
  "actx": {
    "type": "protocol-identifier"
  }
}
```

4.2. JWT Header Requirements

The JWT header for a generic DPoP proof MUST contain the same elements as specified in [RFC9449]:

* typ: MUST be "dpop-proof+jwt"

* alg: An asymmetric signature algorithm identifier

* jwk: The public key used for verification

4.3. JWT Payload Requirements

The JWT payload for a generic DPoP proof MUST contain:

* jti: A unique identifier for the JWT
* iat: Issued-at time
* actx: Authorization Context object (if not using HTTP binding)

Additional claims MAY be included based on context requirements:

* nonce: Server-provided nonce for replay protection
* ath: Access token hash (when presenting access tokens)

5. Protocol Type Registry

This framework establishes a registry for protocol type identifiers used in the actx.type field. Each registered type MUST specify:

1. The additional required and optional fields for the Authorization Context
2. The semantic meaning and validation rules for those fields
3. Security considerations specific to the protocol context
4. Examples of usage

5.1. MOQ Context Type

This section defines the normative authorization context for Media Over QUIC (MOQ) as specified in this framework.

Type Identifier: "moq"

5.1.1. Required Fields

The MOQ authorization context MUST contain the following fields:

action:

A string specifying the MOQ operation (Section 9 of [MOQTransport] being authorized.

tns:

Track Namespace as defined in Section 2.4.1 of [MOQTransport].

5.1.2. Optional Fields

The MOQ authorization context MAY contain the following fields:

tn:

Track Name as defined in Section 2.4.1 of [MOQTransport].

parameters:

An object containing additional MOQ-specific parameters relevant to the operation. The structure and contents of this field are context-dependent.

5.1.3. String Encoding for Binary Data

As defined in Section 2.4.1 of [MOQTransport], Track Namespace is an ordered N-tuple of bytes (where N can be between 1 and 32), and Track Name is a sequence of bytes. Both are not constrained to specific encoding and carry arbitrary byte sequences that are compared by exact byte matching.

For representation in the JSON tns and tn fields of the Authorization Context, the following encoding approach MUST be used:

1. *Track Namespace Tuple Encoding*: The Track Namespace tuple MUST be encoded as a URL-safe string where:
    * Each tuple field is base64url-encoded (without padding)
    * Tuple fields are separated by forward slashes ("/")
    * The entire namespace is prefixed with "moq://"
2. *Track Name Encoding*: The Track Name bytes MUST be base64url-encoded (without padding) when containing non-printable bytes or when exact byte preservation is required

TODO: A better way to encode that promotes readability is being thought about

Example encodings:

* Namespace tuple [0x01, 0x02], [0x03, 0x04]: "moq://AQ/Aw"
* ASCII track name: "camera1"

* Binary track name [0xFF, 0xFE]: "_v4"

* UTF-8 track name with special chars: "m炭sica%20en%20vivo" (URL percent-encoded)

5.1.4. Validation Requirements

Servers processing MOQ authorization contexts MUST:

1. Verify that the action field contains a recognized MOQ operation

2. Validate that the tns field represents a properly formatted track namespace tuple encoding as specified above

3. If present, validate that the tn field contains properly encoded track name bytes

4. Ensure the requested action is permitted for the specified track namespace tuple

5. If present, ensure the requested action is permitted for the specified track name

6. If present, validate any parameters according to usage context

6. Application-Agnostic DPoP Proof JWT Examples

This section provides complete examples of application-agnostic DPoP proof JWTs, illustrating the use of the Authorization Context object in place of HTTP-specific claims.

6.1. Example: MOQ Context DPoP Proof

The following example shows a complete DPoP proof JWT for a MOQ context:

JWT Header:

```
{
  "typ": "dpop-proof+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "x": "l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y": "9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv": "P-256"
  }
}
```

JWT Payload:

```
{
  "jti": "unique-request-id-789",
  "iat": 1705123456,
  "actx": {
    "type": "moq",
    "action": "SUBSCRIBE",
    "tns": "moq://example.com/app/scope/video",
    "tn": "camera1"
  },
  "ath": "fUHyO2r2Z3DZ53EsNrWBb1xWXM4VbCqpW5G-o9GqC7Y"
}
```

6.2. Comparing with HTTP DPoP

For comparison, an equivalent HTTP DPoP proof would contain htm and htu claims instead of the actx object:

```
{
  "jti": "unique-request-id-789",
  "iat": 1705123456,
  "htm": "POST",
  "htu": "https://media.example.com/subscribe",
  "ath": "fUHyO2r2Z3DZ53EsNrWBb1xWXM4VbCqpW5G-o9GqC7Y"
}
```

The key difference is that the application-agnostic framework uses the actx object to provide protocol-specific authorization context, while HTTP DPoP uses htm and htu for HTTP method and URI.

7. Relationship to RFC 9449

This framework extends and generalizes the concepts defined in RFC 9449 [RFC9449] while maintaining full backward compatibility with HTTP-based DPoP implementations.

7.1. Extensions to Section 7 (Protected Resource Access)

RFC 9449 Section 7 defines protected resource access specifically for HTTP contexts. This framework extends those concepts to support non-HTTP protocols while preserving the core security model.

7.1.1. Key Differences from RFC 9449 Section 7

1. *Authorization Context vs HTTP Parameters*: Instead of requiring htm (HTTP method) and htu (HTTP URI) claims, this framework uses the Authorization Context (actx) object to specify protocol-specific request context.

2. *Protocol-Agnostic Token Binding*: While RFC 9449 Section 7.1 defines the DPoP authentication scheme for HTTP Authorization headers, this framework enables token binding for protocols that may not use HTTP-style headers.

3. *Flexible Proof Validation*: The validation rules in RFC 9449 Section 4.3 are adapted to work with the actx object rather than HTTP-specific claims, allowing servers to validate proofs according to their protocol requirements.

7.1.2. Compatibility with RFC 9449

This framework is designed to coexist with RFC 9449 implementations:

* HTTP-based DPoP proofs continue to work unchanged using htm and htu claims

* Generic DPoP proofs use the actx object for non-HTTP contexts

* The same key pairs and token binding mechanisms apply to both approaches

* Authorization servers MAY support both HTTP and generic DPoP proof formats simultaneously

7.1.3. Migration Path

Existing RFC 9449 implementations can adopt this framework incrementally:

1. *HTTP-Only Phase*: Continue using standard RFC 9449 DPoP for HTTP resources

2. *Hybrid Phase*: Support both HTTP DPoP (with htm/htu) and generic DPoP (with actx)

3. *Generic Phase*: Migrate to using Authorization Context objects for all protocols, including HTTP

The typ header value dpop-proof+jwt (instead of dpop+jwt) signals support for the generic framework while maintaining the same cryptographic properties and security model.

8. Security Considerations

This framework inherits all security considerations from [RFC9449]. Additional considerations specific to the generic framework include:

8.1. Context Type Validation

Servers MUST validate that the actx.type field corresponds to a registered and supported context type. Unknown or unsupported context types MUST be rejected.

8.2. Protocol-Specific Security Requirements

Each registered context type MUST specify its own security requirements and threat model. The generic framework does not impose additional security properties beyond those provided by the underlying JWT signature.

8.3. Cross-Context Token Binding

DPoP tokens bound using this framework SHOULD be validated only within their intended context type. Servers MUST NOT accept a DPoP proof with one context type as valid for a different context type.

8.4. Application Separation Requirements

Clients MUST use different DPoP proofs for different applications. This separation ensures that a DPoP proof generated for one application protocol cannot be reused or replayed in the context of another application protocol. Implementations SHOULD enforce this by:

1. *Distinct Key Pairs*: Using separate key pairs for different application protocols where feasible

2. *Context-Specific Binding*: Ensuring that Authorization Context (actx) objects contain fields that uniquely identify the application protocol

3. *Proof Validation*: Rejecting proofs that contain context information from other application protocols

This requirement prevents cross-application attacks where an attacker might attempt to use a valid DPoP proof from one application context in a different application context.

9. IANA Considerations

9.1. DPoP Authorization Context Types Registry

This document establishes the "DPoP Authorization Context Types" registry within the "JSON Web Token (JWT)" registry group.

Registry Name: DPoP Authorization Context Types

Registration Policy: Specification Required

Expert Review Guidelines:

Specifications submitted for registration MUST include:

1. A complete specification of required and optional fields
2. Semantic definitions and validation rules for all fields
3. Security considerations specific to the context type
4. At least one complete example of usage
5. Considerations for interoperability with existing DPoP implementations

Reference Format: The reference MUST be to a published RFC or an Internet-Draft that has been adopted by a working group.

9.1.1. Registration Template

To register a new DPoP Authorization Context Type, the following template MUST be completed:

Type Identifier: The string identifier used in the actx.type field

Description: A brief description of the context type and its intended use

Required Fields: List of mandatory fields in the authorization context object

Optional Fields: List of optional fields in the authorization context object

Validation Rules: Specific validation requirements for the context type

Security Considerations: Context-specific security requirements and threat model

9.1.2. Initial Registry Contents

Type	Description	Required Fields	Optional Fields	Reference
moq	Media Over QUIC authorization context	action, tns	tn, parameters	
RFCXXXX				

Table 1

9.2. JWT Claims Registration

This document registers the following JWT claim in the "JSON Web Token Claims" registry:

Claim Name: actx

Claim Description: Authorization Context Object

Claim Value Type: Object

Change Controller: IETF

Specification Document: This document, Section 4.1

9.3. Media Types Registration

9.3.1. application/dpop-proof+jwt

This document registers the "application/dpop-proof+jwt" media type for generic DPoP proof JWTs in the "Media Types" registry.

Type Name: application

Subtype Name: dpop-proof+jwt

Required Parameters: none

Optional Parameters: none

Encoding Considerations: Binary; base64url encoding of JWT

Security Considerations: See Section 8 of this document

Interoperability Considerations: Multi-Protocol DPoP proofs extend HTTP-specific DPoP while maintaining backward compatibility

Published Specification: This document Applications that use this media type: Applications implementing generic DPoP authorization

Fragment Identifier Considerations: none

Additional Information: none

Person and email address to contact: OAuth Working Group oauth@ietf.org (mailto:oauth@ietf.org)

Intended Usage: COMMON

Restrictions on Usage: none

Author: OAuth Working Group

Change Controller: IETF

10. References

10.1. Normative References

[MOQTransport] Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-13, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-13>.

[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>.

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

[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>.

[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>.

[RFC9449] 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>.

10.2. Informative References

[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>.

Appendix A. Acknowledgments

Thanks for Richard Barnes for the reviews and suggestions on the protocol context design.

Authors' Addresses

Suhas Nandakumar
Cisco Systems
Email: snandaku@cisco.com

Cullen Jennings
Cisco Systems
Email: fluffy@cisco.com