Media Over QUIC
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2026

S. Nandakumar
C. Jennings
Cisco Systems

1 March 2026

Application-Agnostic Demonstration Proof of Possession (DPoP) Framework
draft-nandakumar-moq-generic-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. The framework supports both JWT-based proofs (for compatibility with existing OAuth deployments) and CWT-based proofs (for compact binary encoding and interoperability with Common Access Token systems).

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

Discussion of this document takes place on the Media Over QUIC Working Group mailing list (mailto:moq@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/moq/. Subscribe at https://www.ietf.org/mailman/listinfo/moq/.

Source for this draft and an issue tracker can be found at https://github.com/moq-wg/dpop-generic-proof.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 September 2026.

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].

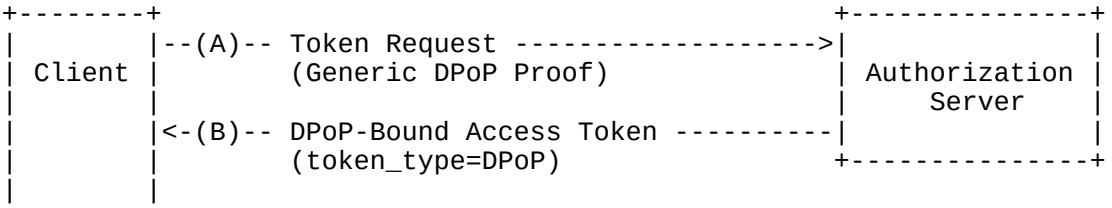
The terms "CBOR Web Token (CWT)", "COSE", and "COSE Key" are defined in [RFC8392], [RFC9052], and [RFC9053].

The term "CWT Confirmation" is defined in [RFC8747].

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.



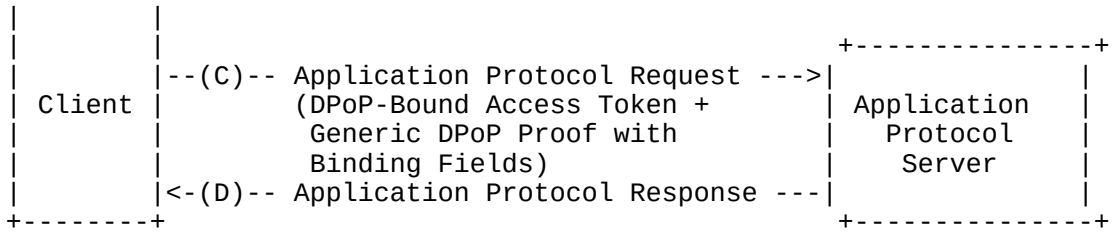


Figure 1: Generic DPoP Flow

The main data structure introduced by this specification is a generic DPoP proof that can be used across various application protocols and contexts, as described in detail below. This proof can be encoded as either a JWT (JSON Web Token) or CWT (CBOR Web Token), depending on deployment requirements. A client uses a generic DPoP proof 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 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 framework supports DPoP proofs in two formats:

1. *JWT-based DPoP Proofs*: Using JSON Web Token structure as defined in [RFC7519], consistent with RFC 9449

2. *CWT-based DPoP Proofs*: Using CBOR Web Token structure as defined in [RFC8392], enabling compact binary encoding suitable for constrained environments and interoperability with Common Access Token (CAT) [CTA-5007-B]

Both formats share the same core 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. Choosing Between JWT and CWT Formats

Implementations should choose between JWT and CWT formats based on their deployment requirements:

Use JWT-based DPoP proofs when:

* Integrating with existing OAuth 2.0 infrastructure that uses JWT

* Interoperating with systems that expect JSON-based tokens

* Human readability of tokens is beneficial for debugging

* The deployment does not have strict size constraints

Use CWT-based DPoP proofs when:

* Integrating with Common Access Token (CAT) [CTA-5007-B] systems

* Operating in bandwidth-constrained environments where compact encoding matters

* The protocol already uses CBOR encoding for other data structures

* Interoperating with IoT or constrained device deployments

Servers MAY support both formats simultaneously. When doing so, they MUST validate the format indicator in the proof header (dpop-proof+jwt or dpop-proof+cwt) and process the proof according to the indicated format.

4.2. 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.

4.2.1. 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"
  }
}
```

4.3. JWT-based DPoP Proof Structure

This section defines the JWT-based DPoP proof format for use with JSON-based systems and standard OAuth deployments.

4.3.1. 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

4.3.2. 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:

* ath: Access token hash. REQUIRED when the DPoP proof is sent together with an access token to a resource server. Contains the base64url-encoded SHA-256 hash of the access token.

* nonce: Server-provided nonce for replay protection. OPTIONAL,

included when the server has provided a nonce value.

4.4. CWT-based DPoP Proof Structure

This section defines the CWT-based DPoP proof format for use with CBOR-based systems, constrained environments, and systems using Common Access Token (CAT) [CTA-5007-B].

4.4.1. CWT Protected Header Requirements

The CWT protected header for a generic DPoP proof MUST contain the following parameters defined in [RFC9052]:

* alg (label 1): A COSE algorithm identifier for an asymmetric signature algorithm (e.g., -7 for ES256, -35 for ES384, -36 for ES512)

* typ (label 16): MUST be "dpop-proof+cwt"

* COSE_Key (label 4): The public key used for verification, encoded as a COSE_Key structure per [RFC9052]

4.4.2. CWT Claims Requirements

The CWT claims set for a generic DPoP proof MUST contain:

* cti (label 7): A unique identifier for the CWT, encoded as a byte string

* iat (label 6): Issued-at time as a NumericDate

* actx (label TBD): Authorization Context object

Additional claims:

* ath (label TBD): Access token hash. REQUIRED when the DPoP proof is sent together with an access token to a resource server. Encoded as a byte string containing the SHA-256 hash of the access token.

* nonce (label TBD): Server-provided nonce for replay protection. OPTIONAL, included when the server has provided a nonce value. Encoded as a text string.

4.4.3. CWT Authorization Context Structure

The Authorization Context in CWT format uses CBOR encoding with integer keys for compact representation:

```
actx = {
  type: tstr,           ; Protocol type identifier (key 0)
  * int => any          ; Protocol-specific fields
}
```

For MOQT contexts, the CWT Authorization Context uses:

```
moqt-actx = {
  0: "moqt",            ; type
  1: tstr,              ; action
  2: tstr,              ; tns (track namespace)
  ? 3: tstr,            ; tn (track name)
  ? 4: any              ; parameters
}
```

4.4.4. CDDL Definition for CWT DPoP Proof

The complete CDDL definition for a CWT-based DPoP proof is:

```
dpop-proof-cwt = COSE_Sign1

dpop-protected-header = {
  1 => int,                    ; alg
  16 => "dpop-proof+cwt",      ; typ
  4 => COSE_Key                ; COSE_Key
}

; Claims for DPoP proof at token endpoint (no access token yet)
dpop-cwt-claims-token-request = {
  7 => bstr,                   ; cti (unique identifier)
  6 => numericdate,            ; iat (issued at)
  actx-label => actx,          ; actx (authorization context)
  ? nonce-label => tstr,       ; nonce (optional)
}

; Claims for DPoP proof at resource server (with access token)
dpop-cwt-claims-resource-access = {
  7 => bstr,                   ; cti (unique identifier)
  6 => numericdate,            ; iat (issued at)
  actx-label => actx,          ; actx (authorization context)
  ath-label => bstr,           ; ath (REQUIRED, SHA-256 hash of access token)
  ? nonce-label => tstr,       ; nonce (optional)
}

actx-label = TBD              ; To be assigned by IANA
nonce-label = TBD             ; To be assigned by IANA
ath-label = TBD               ; To be assigned by IANA

numericdate = int / float
```

4.4.5. Token Binding with CWT Access Tokens

When using CWT-based access tokens (such as Common Access Token), the DPoP binding is established using the confirmation claim (cnf, label 8) as defined in [RFC8747]. The cnf claim contains the key confirmation:

* For JWT DPoP proofs: Use the jkt confirmation method containing the JWK SHA-256 Thumbprint as defined in [RFC7638]

* For CWT DPoP proofs: Use the ckt confirmation method containing the COSE Key Thumbprint as defined in [RFC9679]

Example CWT access token with DPoP binding:

```
{
  / iss / 1: "auth.example.com",
  / aud / 3: "resource.example.com",
  / exp / 4: 1705209856,
  / iat / 6: 1705123456,
  / cnf / 8: {
    / jkt / 323: h'fUHyO2r2Z3DZ53EsNrWBb1xWXM4VbCqpW5G-o9GqC7Y'
  }
}
```

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. MOQT Context Type

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

Type Identifier: "moqt"

5.1.1. Required Fields

The MOQT authorization context MUST contain the following fields:

action:

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

tns:

Track Namespace tuple serialized using the canonical encoding format defined in Section 5.1.

5.1.2. Optional Fields

The MOQT authorization context MAY contain the following fields:

tn:

Track Name serialized using the canonical encoding format defined in Section 5.1.

parameters:

An object containing additional MOQT-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 and Track Name is a sequence of bytes.

The tns and tn fields in the Authorization Context MUST use the canonical serialization format defined in Section 1.5.1 of [MOQTransport]:

* The tns field contains namespace tuple elements, each serialized per Section 1.5.1, joined by hyphens (-)

* The tn field (when present) contains the track name serialized per Section 1.5.1

5.1.3.1. Examples

* Namespace ("example.net", "team2", "project_x") with track name "report": "tns": "example.2enet-team2-project_x", "tn": "report"

* Namespace ("conference", "room1") with track name "audio.opus": "tns": "conference-room1", "tn": "audio.2eopus"

* Namespace with binary bytes [0xFF, 0x01] and [0x02]: "tns":

".ff.01-.02"

5.1.4. Validation Requirements

Servers processing MOQ authorization contexts MUST:

1. Verify that the action field contains a recognized MOQT 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 Examples

This section provides complete examples of application-agnostic DPoP proofs in both JWT and CWT formats, illustrating the use of the Authorization Context object in place of HTTP-specific claims.

6.1. JWT Format Examples

6.1.1. Example: MOQT Context DPoP Proof (JWT)

The following example shows a complete DPoP proof JWT for a MOQT 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": "moqt",
    "action": "SUBSCRIBE",
    "tns": "example.2ecom-app-scope-video",
    "tn": "camera1"
  },
  "ath": "fUHyO2r2Z3DZ53EsNrWBb1xWXM4VbCqpW5G-o9GqC7Y"
}
```

6.1.2. 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.

6.2. CWT Format Examples

6.2.1. Example: MOQT Context DPoP Proof (CWT)

The following example shows a complete DPoP proof CWT for a MOQT context using CBOR diagnostic notation. This is the proof that the client creates and sends to demonstrate possession of the private key corresponding to the public key bound to the access token:

CWT Protected Header:

```
{
  / alg: ES256 / 1: -7,
  / typ / 16: "dpop-proof+cwt",
  / COSE_Key / 4: {
    / kty: EC2 / 1: 2,
    / crv: P-256 / -1: 1,
    / x / -2: h'97cb45ae1c7f37855d788810883698f730a40e5a4194
                0e24502bbf915a56606b',
    / y / -3: h'f55138a5ff13ca4f7e06b4d2cbcd0c1697f3de37c6d4
                a6eb5735ebd5fc01483f'
  }
}
```

CWT Claims Set:

```
{
  / cti / 7: h'756e697175652d72657175657374',
  / iat / 6: 1705123456,
  / actx / TBD: {
    / type / 0: "moqt",
    / action / 1: "SUBSCRIBE",
    / tns / 2: "example.2ecom-app-scope-video",
    / tn / 3: "camera1"
  },
  / ath / TBD: h'7d41f23b6af667701de7712c36b5816f5c5619
                cc555cca63ab099fa8f46a8ca6'
}
```

6.2.2. Example: CWT DPoP Proof with Common Access Token

This example shows the structure of the access token issued by the authorization server, not the DPoP proof itself. When using CWT DPoP proofs with Common Access Token (CAT) [CTA-5007-B], the access token is a CWT with a cnf claim binding to the DPoP proof key.

Access Token (CWT):

```
{
  / iss / 1: "auth.example.com",
  / aud / 3: "resource.example.com",
  / exp / 4: 1705209856,
  / iat / 6: 1705123456,
```

```
  / cti / 7: h'746f6b656e2d6964',
  / cnf / 8: {
    / jkt / 323: h'7d41f23b6af667701de7712c36b5816f5c5619
                  cc555cca63ab099fa8f46a8ca6'
  },
  / catdpop / 321: {
    / window / 0: 300,
    / jti / 1: 1
  }
}
```

The catdpop claim (label 321) provides DPoP-specific settings as defined in [CTA-5007-B]:

* window (key 0): Acceptable time window for DPoP proofs in seconds

* jti (key 1): JTI processing semantics (0=ignore, 1=may honor for replay detection)

The corresponding DPoP proof binds to this access token through the ath claim containing the SHA-256 hash of the access token.

6.2.3. Comparing JWT and CWT Formats

The following table summarizes the mapping between JWT and CWT DPoP proof elements:

JWT Element	CWT Element	Description
Header.typ	Protected.typ (16)	"dpop-proof+jwt" or "dpop-proof+cwt"
Header.alg	Protected.alg (1)	Signature algorithm
Header.jwk	Protected.COSE_Key (4)	Public key for verification
Payload.jti	Claims.cti (7)	Unique identifier
Payload.iat	Claims.iat (6)	Issued-at timestamp
Payload.actx	Claims.actx (TBD)	Authorization context
Payload.nonce	Claims.nonce (TBD)	Server-provided nonce
Payload.ath	Claims.ath (TBD)	Access token hash

Table 1

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.

8.5. CWT-Specific Security Considerations

When using CWT-based DPoP proofs, the following additional considerations apply:

1. *Algorithm Selection*: CWT DPoP proofs MUST use COSE algorithms registered for use with COSE_Sign1 as defined in [RFC9053]. The same algorithm restrictions from [RFC9449] apply: symmetric algorithms MUST NOT be used.

2. *Key Confirmation Methods*: When binding CWT access tokens to DPoP proofs, implementations SHOULD prefer the jkt confirmation method for interoperability with JWT-based DPoP proofs, or ckt (COSE Key Thumbprint) when both token and proof are CWT-based.

3. *Binary Encoding*: While CWT provides compact encoding, implementations MUST ensure that the encoded proof does not exceed transport-specific size limits.

4. *Cross-Format Attacks*: Servers that accept both JWT and CWT DPoP proofs MUST validate the format indicator (dpop-proof+jwt vs dpop-proof+cwt) and reject proofs where the format indicator does not match the actual encoding.

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
moqt	Media Over QUIC Transport authorization context	action, tns	tn, parameters	
RFCXXXX				

Table 2

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 4.3

9.3. CWT Claims Registration

This document registers the following claims in the "CBOR Web Token (CWT) Claims" registry defined in [RFC8392]:

9.3.1. actx Claim

Claim Name: actx

Claim Description: Authorization Context Object for DPoP proofs

JWT Claim Name: actx

Claim Key: TBD (requested: 400)

Claim Value Type: map

Change Controller: IETF

Specification Document: This document, Section 4.4

9.3.2. nonce Claim (DPoP)

Claim Name: dpop_nonce

Claim Description: Server-provided nonce for DPoP replay protection

JWT Claim Name: nonce

Claim Key: TBD (requested: 401)

Claim Value Type: text string

Change Controller: IETF

Specification Document: This document, Section 4.4

9.3.3. ath Claim

Claim Name: ath

Claim Description: Access Token Hash for DPoP proof binding

JWT Claim Name: ath

Claim Key: TBD (requested: 402)

Claim Value Type: byte string

Change Controller: IETF

Specification Document: This document, Section 4.4

9.4. Media Types Registration

9.4.1. 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: Media Over QUIC Working Group moq@ietf.org (mailto:moq@ietf.org)

Intended Usage: COMMON

Restrictions on Usage: none

Author: Media Over QUIC Working Group

Change Controller: IETF

9.4.2. application/dpop-proof+cwt

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

Type Name: application

Subtype Name: dpop-proof+cwt

Required Parameters: none

Optional Parameters: none

Encoding Considerations: Binary; CBOR encoding as defined in [RFC8949]

Security Considerations: See Section 8 of this document

Interoperability Considerations: CWT-based DPoP proofs provide compact binary encoding suitable for constrained environments and interoperability with Common Access Token (CAT) systems

Published Specification: This document

Applications that use this media type: Applications implementing generic DPoP authorization with CBOR/CWT-based tokens

Fragment Identifier Considerations: none

Additional Information: none

Person and email address to contact: Media Over QUIC Working Group moq@ietf.org (mailto:moq@ietf.org)

Intended Usage: COMMON

Restrictions on Usage: none

Author: Media Over QUIC 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-16, 13 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-16>.

[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>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/rfc/rfc7638>.

[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>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/rfc/rfc8747>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[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>.

[RFC9679] Isobe, K., Tschofenig, H., and O. Steele, "CBOR Object Signing and Encryption (COSE) Key Thumbprint", RFC 9679, DOI 10.17487/RFC9679, December 2024, <https://www.rfc-editor.org/rfc/rfc9679>.

10.2. Informative References

[CTA-5007-B]
Consumer Technology Association, "Common Access Token", April 2025, <https://shop.cta.tech/products/common-access-token>.

[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