

Media Over QUIC
Internet-Draft
Intended status: Informational
Expires: 23 March 2026

W. Law
Akamai
C. Lemmons
Comcast
G. Simon
Synamedia
S. Nandakumar
Cisco
19 September 2025

Authentication scheme for MOQT using Common Access Tokens
draft-ietf-moq-c4m-00

Abstract

A token-based authentication scheme for use with Media Over QUIC Transport.

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/CAT-4-MOQT/>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-moq-c4m/>.

Discussion of this document takes place on the Media Over QUIC 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/CAT-4-MOQT>.

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 23 March 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Overview of the authentication workflow	3
2. Token format	4
2.1. moqt claim	5
2.1.1. Text examples of permissions to help with CDDL construction	6
2.1.2. Multiple actions	8
2.2. moqt-reval claim	9
3. DPoP Integration with CAT for MOQT	10
3.1. CAT DPoP Claims for MOQT	10
3.1.1. Confirmation (cnf) Claim with JWK Thumbprint	10
3.1.2. DPoP Extension with Application-Agnostic Proof Framework	11
3.1.3. MOQT Resource URI Construction	14
3.2. DPoP Proof Process and Token Binding Flow	14
3.2.1. Phase 1: Token Acquisition with DPoP Binding	14
3.2.2. Phase 2: MOQT Operations with DPoP Proof Validation	16
4. Adding a token to a URL	18
5. Conventions and Definitions	18
6. Security Considerations	18
7. IANA Considerations	18
8. Normative References	19
Acknowledgments	20
Authors' Addresses	20

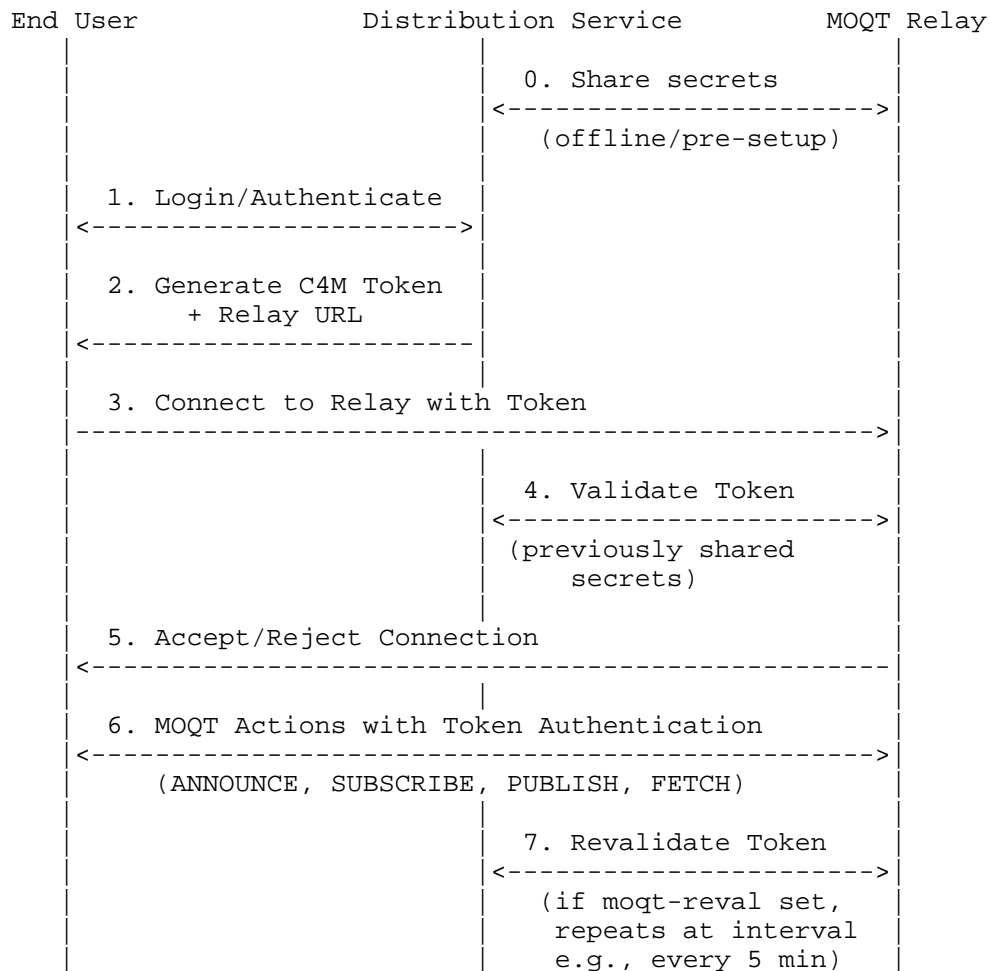
1. Introduction

This draft introduces a token-based authentication scheme for use with MOQT [MoQTransport]. The scheme protects access to the relay during session establishment and also constrains the actions which the client may take once connected.

This draft defines version 1 of this specification.

1.1. Overview of the authentication workflow

- * An end-user logs-in to a distribution service. The service authenticates the user (via username/password, OAuth, 2FA or another method). The methods involved in this authentication step lie outside the scope of this draft.
- * Based upon the identity and permissions granted to that end-user, the service generates a token. A token is a data structure that has been serialized into a byte array. The token encodes information such as the user's ID, constraints on how and when they can access the MOQT distribution network and constraints on the actions they can take once connected. The token may be signed to make it tamper-resistant.
- * The token is given in the clear to the end-user, along with a URL to connect to the edge relay of a MOQT distribution network. The edge relay is part of a trusted MOQT distribution network. It has previously shared secrets with the distribution service, so that this relay is entitled to decrypt related tokens and to validate signatures.
- * The end-user client application provides the token to the MOQT distribution relay when it connects. This connection may be established over WebTransport or raw QUIC.
- * The relay decrypts the token upon receipt and validates the signature. Based upon claims conveyed in the token, the relay accepts or rejects the connection.
- * If the relay accepts the connection, then the client will take a series of MOQT actions: ANNOUNCE, SUBSCRIBE_ANNOUNCES, SUBSCRIBE or FETCH. For each of these, it will supply the token it received using the AUTHENTICATION parameter.
- * As an alternative to this workflow, the distribution service may vend multiple tokens to the client. The client may use one of those tokens to establish the initial connection and others to authenticate its actions.



2. Token format

This draft uses a single token format, namely the Common Access Token (CAT) [CAT]. The token is supplied as a byte array. When it must be cast to a string for inclusion in a URL, it is Base64 encoded [BASE64].

To provide control over the MOQT actions, this draft defines a new CBOR Web Token (CWT) Claim called "moqt". Use of the moqt claim is optional for clients. Support for processing the moqt claim is mandatory for relays.

The default for all actions is "Blocked" and this does not need to be communicated in the token. As soon as a token is provided, all actions are explicitly blocked unless explicitly enabled.

2.1. moqt claim

The "moqt" claim is defined by the following CDDL:

```
$$Claims-Set-Claims /= (moqt-label => moqt-value)
moqt-label = TBD_MOQT
moqt-value = [ + moqt-scope ]
moqt-scope = [ moqt-actions, moqt-ns-match, moqt-track-match ]
moqt-actions = [ + moqt-action ]
moqt-action = int
moqt-ns-match = bin-match
moqt-track-match = bin-match

bin-match = {
  ? exact-match ^ => bstr,
  ? prefix-match ^ => bstr,
  ? suffix-match ^ => bstr,
  ? contains-match ^ => bstr,
}

/ match labels defined in CTA-5007-B 4.6.1 /
exact-match = 0
prefix-match = 1
suffix-match = 2
contains-match = 3
```

The "moqt" claim bounds the scope of MOQT actions for which the token can provide access. It is an array of action scopes. Each scope is an array with three elements: an array of integers that identifies the actions, a match object for the namespace, and a match object for the track name.

The actions are integers defined as follows:

Action	Key	Reference
CLIENT_SETUP	0	[MoQTransport] Section 8.3
SERVER_SETUP	1	[MoQTransport] Section 8.3
ANNOUNCE	2	[MoQTransport] Section 8.23
SUBSCRIBE_NAMESPACE	3	[MoQTransport] Section 8.28
SUBSCRIBE	4	[MoQTransport] Section 8.7
SUBSCRIBE_UPDATE	5	[MoQTransport] Section 8.10
PUBLISH	6	[MoQTransport] Section 8.13
FETCH	7	[MoQTransport] Section 8.16
TRACK_STATUS	8	[MoQTransport] Section 8.20

Table 1

The scope of the moqt claim is limited to the actions provided in the array. Any action not present in the array is not authorized by moqt claim.

The match object is defined to be a binary form of the match object defined in [CAT] Section 4.6.1. The regex and hash match types are not defined for use with binary values in this document.

The first match operation is performed against the namespace and the second against the track name (as defined in Section 2.4.1 of {draft-ietf-moq-transport}). Since the match is not being performed against a URI, no normalization is performed and the matches are performed against the entire string. An empty match object is a legal construct that matches all names.

2.1.1. Text examples of permissions to help with CDDL construction

Example: Allow with an exact match "example.com/bob"

```
{
  /moqt/ TBD_MOQT: [[
    [ /ANNOUNCE/ 2, /SUBSCRIBE_NAMESPACE/ 3, /PUBLISH/ 6, /FETCH/ 7 ],
    { /exact/ 0: 'example.com' },
    { /exact/ 0: '/bob' }
  ]]
}
```

Permits

- * 'example.com', '/bob'

Prohibits

- * 'example.com', ''
- * 'example.com', '/bob/123'
- * 'example.com', '/alice'
- * 'example.com', '/bob/logs'
- * 'alternate/example.com', '/bob'
- * '12345', ''
- * 'example', '.com/bob'

Example: Allow with a prefix match "example.com/bob"

```
{
  /moqt/ TBD_MOQT: [[
    [ /ANNOUNCE/ 2, /SUBSCRIBE_NAMESPACE/ 3, /PUBLISH/ 6, /FETCH/ 7 ],
    { /exact/ 0: 'example.com' },
    { /prefix/ 1: '/bob' }
  ]]
}
```

Permits

- * 'example.com', '/bob'
- * 'example.com', '/bob/123'
- * 'example.com', '/bob/logs'

Prohibits

- * 'example.com', ''
- * 'example.com', '/alice'
- * 'alternate/example.com', '/bob'
- * '12345', ''
- * 'example', '.com/bob'

2.1.2. Multiple actions

Multiple actions may be communicated within the same token, with different permissions. This can be facilitated by the logical claims defined in [Composite] or simply by defining multiple limits, depending on the required restrictions. In both cases, the order in which limits are declared and evaluated is unimportant. The evaluation stops after the first acceptable result is discovered.

2.1.2.1. Example of evaluating multiple actions in the same token:

```
{
  /moqt/ TBD_MOQT: [
    [/PUBLISH/ 6, { /exact/ 0: 'example.com'}, { /prefix/ 1: 'bob'}],
    [/PUBLISH/ 6, { /exact/ 0: 'example.com'}, { /exact/ 0: 'logs/12345/bob'}]
  ],
  /exp/ 4: 1750000000
}
```

* (1) PUBLISH (Allow with a prefix match) example.com/bob

* (2) PUBLISH (Allow with an exact match) example.com/logs/12345/bob

Evaluating "example.com/bob/123" would succeed on test 1 and test 2 would never be evaluated. Evaluating "example.com/logs/12345/bob" would fail on test 1 but then succeed on test 2. Evaluating "example.com" would fail on test 1 and on test 2.

In addition, the entire token expires at 2025-05-02T21:57:24+00:00.

2.1.2.2. Example of evaluating multiple actions with related claims:

If there are other claims that depend on which MOQT limit applies, a logical claim is required:

```
{
  /or/ TBD_OR: [
    {
      /moqt/ TBD_MOQT: [[/PUBLISH/ 6, { /exact/ 0: 'example.com'}, { /prefix/ 1: 'bob'}]],
      /exp/ 4: 1750000000
    },
    {
      /moqt/ TBD_MOQT: [[/PUBLISH/ 6, { /exact/ 0: 'example.com'}, { /exact/ 0: 'logs/12345/bob'}]],
      /exp/ 4: 1750000600
    }
  ]
}
```

This provides access to the same tracks as the previous example, but in this case, the token is valid for publishing logs up to 10 minutes after the time at which the publishing of the bob track expires.

DISCUSS: Because tokens are designed for instantaneous evaluation, they naturally only evaluate to an "acceptable" or an "unacceptable". It's somewhat tricky to turn an evaluation into a complete bound on any particular value. The CAT has a number of claims about the context of the request that can change while the stream is open. The most obvious of these is the expiration time. The "catnip" (Network IP) and geographic claims can also change mid-stream if the connection is migrated or the client moves. Do we need to do something special to require periodic re-evaluation?

2.2. moqt-reval claim

The "moqt-reval" claim is defined by the following CDDL:

```
$$Claims-Set-Claims ::= (moqt-reval-label => moqt-reval-value)
moqt-reval-label = TBD_MOQT_REVAL
moqt-reval-value = number
```

The "moqt-reval" claim indicates that the token must be revalidated for ongoing streams. If the token is no longer acceptable, the actions authorized by it MUST not be permitted to continue.

The "moqt-reval-value" is a revalidation interval, expressed in seconds. It provides an upper bound on how long a token may be considered acceptable for an ongoing stream. A revalidator MAY revalidate sooner.

If the revalidation interval is smaller than the recipient is prepared or able to revalidate, the recipient MUST reject the token. If a recipient is unable to revalidate tokens, it MUST reject all tokens with a "moqt-reval" claim.

A token can be revalidated by simply validating it again, just as if it were new. However, since some claims, signatures, MACs, and other attributes that could contribute to unacceptability may be incapable of changing acceptability in the duration, a revalidator may optimize by skipping some of the checks as long as the outcome of the validation is the same. Revalidators SHOULD skip reverifying MACs and signatures when the list of acceptable issuer keys is unchanged.

When the value of this claim is zero, the token MUST NOT be revalidated. This is the default behaviour when the claim is not present.

This claim MUST NOT be used outside of a base claimset. If used within a composition claims, the token is not well-formed.

The claim key for this claim is TBD_MOQT_REVAL and the claim value is a number. Recipients MUST support this claim. This claim is OPTIONAL for issuers.

3. DPoP Integration with CAT for MOQT

This section defines the use of CAT's Demonstrating Proof of Possession (DPoP) claims [DPoP] to enhance security in MOQT environments. This approach leverages the CAT token's "cnf" (confirmation) claim with JWK Thumbprint binding and the "catdpop" (CAT DPoP Settings) claim to provide proof-of-possession capabilities that prevent token theft and replay attacks in MOQT systems.

3.1. CAT DPoP Claims for MOQT

This proposal extends the CAT authentication model by binding tokens to client cryptographic key pairs. To enable sender-constrained token usage, the CAT tokens include DPoP-related claims as defined [CAT] Section 4.8, ensuring that only the legitimate token holder can use the token for MOQT operations.

3.1.1. Confirmation (cnf) Claim with JWK Thumbprint

DPoP binding is accomplished by providing the "cnf" claim with the "jkt" (JWK Thumbprint) confirmation method.

Below is an example showing jkt token binding.

```
{
  / cnf /
  8: {
    / jkt /
    3: <32-byte JWK SHA-256 Thumbprint>
  },
  / moqt /
  TBD_MOQT: [
    [
      [2, 3, 6, 7], / ANNOUNCE, SUBSCRIBE_NAMESPACE, PUBLISH, FETCH /
      {"exact": "cdn.example.com"},
      {"prefix": "/sports/"}
    ]
  ],
  / catdpop /
  321: {
    0: 300, / 5-minute window /
    1: 1 / Honor jti for replay protection /
  },
  / exp /
  4: 1750000000
}
```

Implementation Requirements:

- * Relay Validation: MOQT relays MUST verify that DPoP proofs are signed with the private key corresponding to the "jkt" value
- * Proof Binding: Relays MUST reject requests where DPoP proof validation or key binding fails
- * Processing Semantics: Relays MUST process DPoP proofs as Protected Resource Access requests per [DPoP] Section 7

3.1.2. DPoP Extension with Application-Agnostic Proof Framework

This section defines the use of DPoP with an application-agnostic proof framework as specified in [DPOP-PROOF], which extends the traditional HTTP-centric DPoP model to support arbitrary protocols including MOQT. This approach replaces HTTP-specific claims with a flexible authorization context structure that can accommodate protocol-specific command representations.

The DPoP proof JWT follows the structure defined in Section 4 of [DPOP-PROOF] with the following required claims:

JWT Header:

- * "typ": "dpop-proof+jwt"
- * "alg": Asymmetric signature algorithm identifier
- * "jwk": Public key for verification

JWT Payload:

- * "jti": Unique identifier for the JWT
- * "iat": Issued-at time
- * "actx": Authorization Context object

For MOQT operations, the Authorization Context ("actx") object contains:

- * "type": "moqt" (registered identifier for MOQT protocol)
- * "action": MOQT action identifier
- * "tns": Track namespace (required)
- * "tn": Track name (required)
- * "resource": MOQT resource identifier (optional)

When the optional "resource" parameter is included, it MUST be consistent with the "tns" and "tn" parameters. The resource URI should follow the format `moqt://<relay-endpoint>?tns=<namespace>&tn=<track>` where the tns and tn query parameters match the respective "tns" and "tn" fields in the Authorization Context.

Example DPoP proof for MOQT ANNOUNCE operation:

```

{
  "typ": "dpop-proof+jwt",
  "alg": "ES256",
  "jwk": { ... }
}
.
{
  "jti": "unique-request-id",
  "iat": 1705123456,
  "actx": {
    "type": "moqt",
    "action": "ANNOUNCE",
    "tns": "sports",
    "tn": "live-feed"
  }
}

```

MOQT action mapping for Authorization Context:

MOQT Action	actx.action
CLIENT_SETUP	SETUP
SERVER_SETUP	SETUP
ANNOUNCE	ANNOUNCE
SUBSCRIBE_NAMESPACE	SUB_NS
SUBSCRIBE	SUBSCRIBE
PUBLISH	PUBLISH
FETCH	FETCH

Table 2

Relays supporting this application-agnostic DPoP framework MUST:

- * Validate DPoP proofs according to [DPOP-PROOF]
- * Verify that the "actx.type" is "moqt" for MOQT operations
- * Validate that the "actx.action" matches the requested MOQT action

- * Verify that the "actx.tns" corresponds to the target track namespace
- * Verify that the "actx.tn" corresponds to the target track name
- * If present, verify the "actx.resource" is consistent with "tns" and "tn"
- * Reject requests where Authorization Context validation fails

3.1.3. MOQT Resource URI Construction

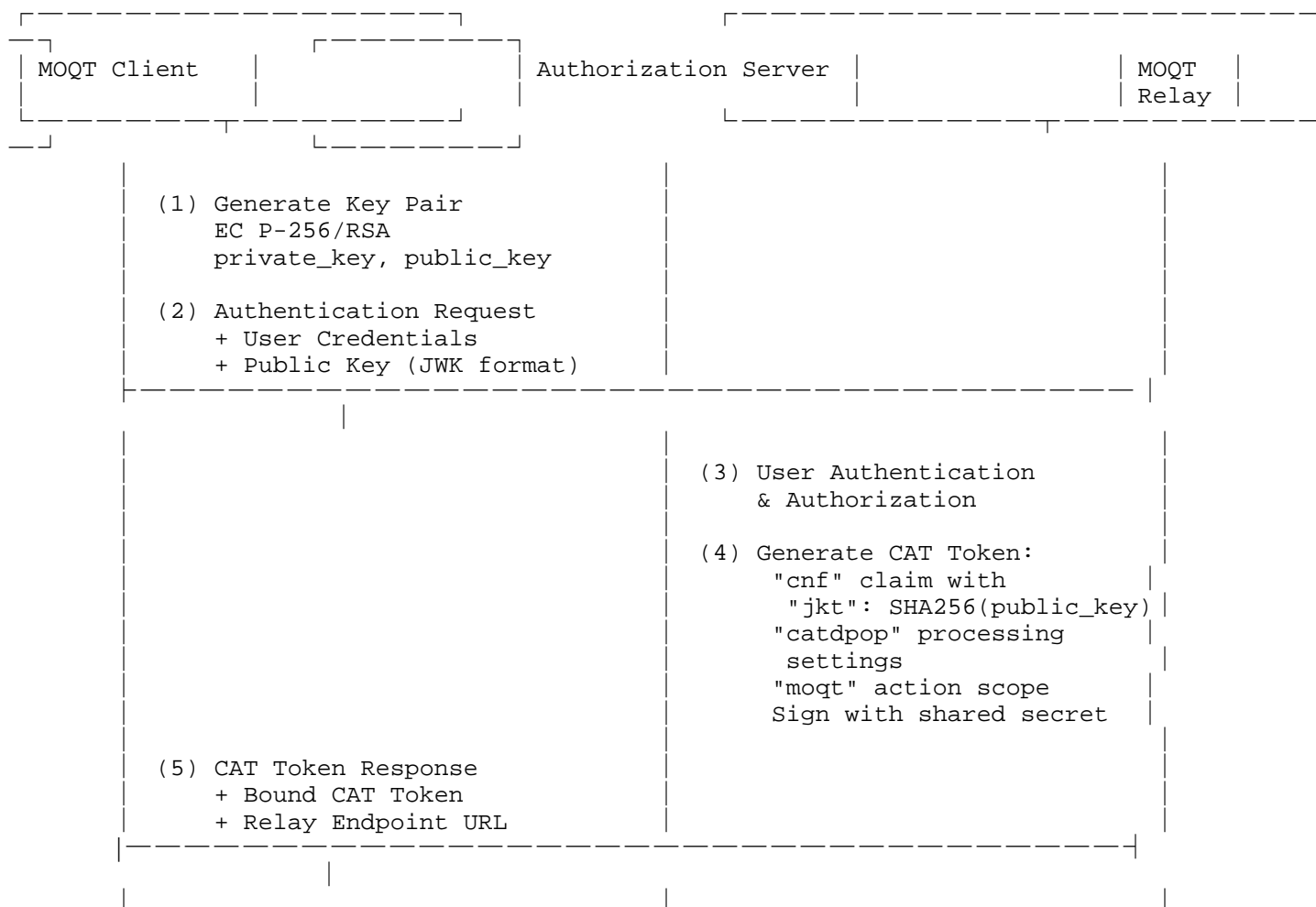
The Authorization Context "resource" field should specify track namespace (tns) and track name (tn) parameters for MOQT resources:

- * Connection setup: moqt://<relay-endpoint>
- * Namespace operations: moqt://<relay-endpoint>?tns=<namespace>
- * Track operations: moqt://<relay-endpoint>?tns=<namespace>&tn=<track>

3.2. DPoP Proof Process and Token Binding Flow

The following process illustrates how DPoP proof provision results in CAT token binding and subsequent MOQT relay validation:

3.2.1. Phase 1: Token Acquisition with DPoP Binding



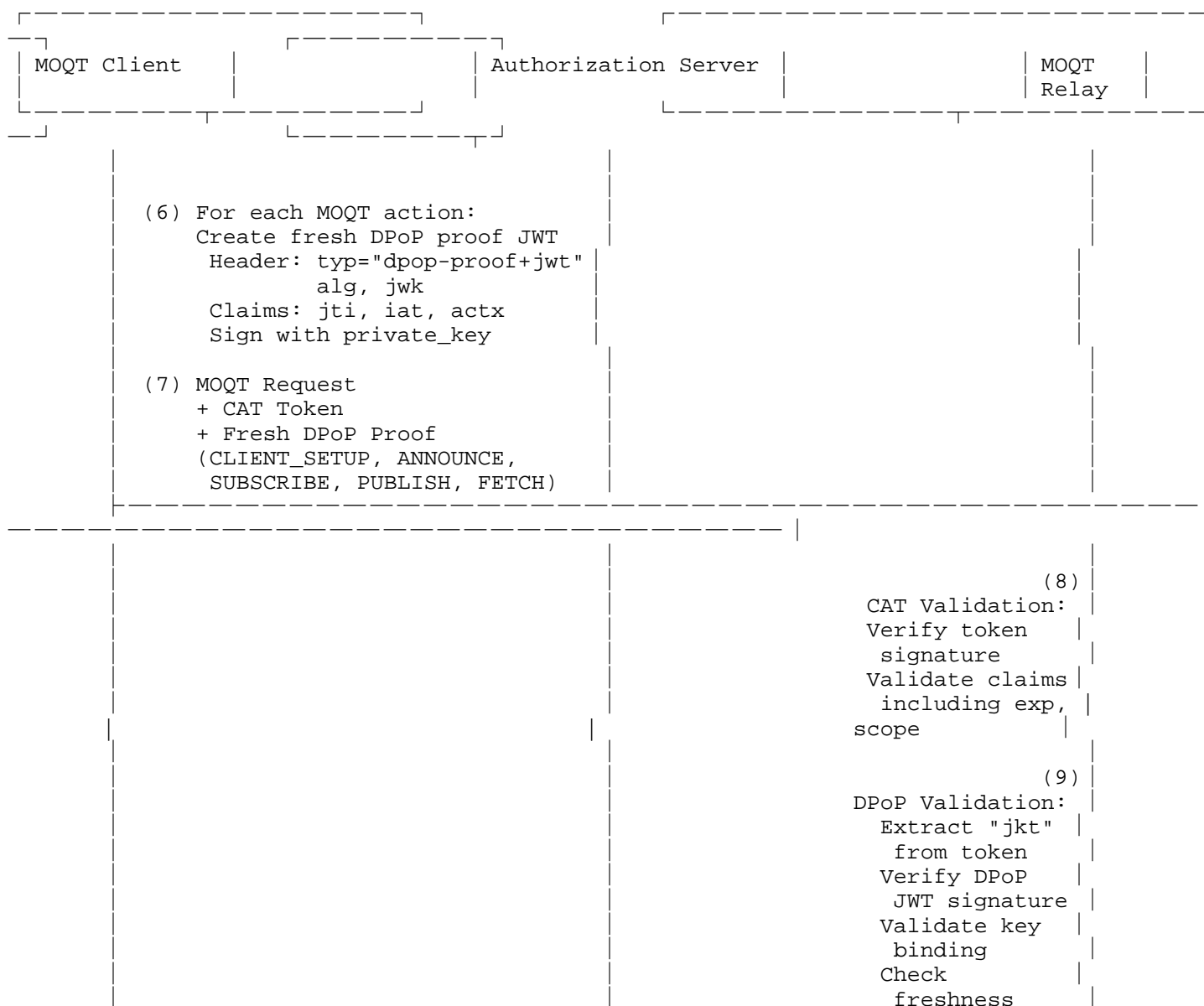
Steps 1-5 Detail:

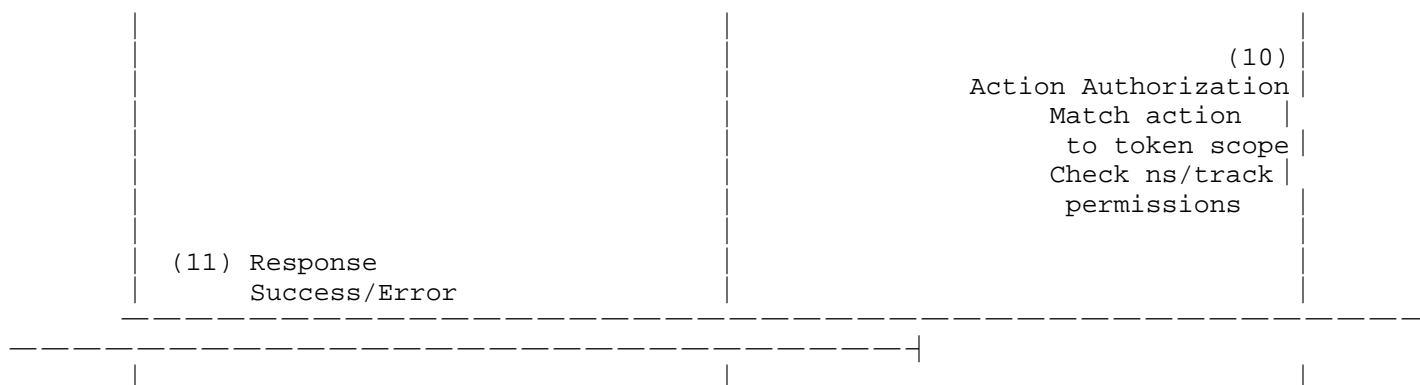
1. **Client Key Generation:** The MOQT client generates an asymmetric key pair (typically EC P-256) for DPoP operations
2. **Authentication with Public Key:** Client authenticates with the authorization server, providing user credentials and the public key
3. **User Authentication:** Authorization server validates user identity and permissions
4. **CAT Token Generation:** Server creates a CAT token containing:
 - * "cnf" claim: JWK Thumbprint ("jkt") of the client's public key (32-byte SHA-256 hash)

- * "catdpop" claim: DPoP processing settings (window, jti handling, critical settings)
- * "moqt" claim: Authorized MOQT actions and scope restrictions

5. Token Delivery: Server provides the bound CAT token and relay endpoint information to the client

3.2.2. Phase 2: MOQT Operations with DPoP Proof Validation





Steps 6-11 Detail:

1. DPoP Proof Creation: For each MOQT action, the client creates a fresh DPoP proof JWT with:
 - * Header: typ: "dpop-proof+jwt", alg, jwk (public key)
 - * Claims: jti (unique ID), iat (timestamp), actx (Authorization Context with type, action, tns, tn)
2. MOQT Request: Client sends MOQT action with both CAT token and fresh DPoP proof
3. CAT Token Validation: Relay validates:
 - * Token signature using shared secret with authorization server
 - * Token expiration time
 - * "moqt" claim scope for requested action
4. DPoP Proof Validation: Relay performs:
 - * Extract "jkt" (JWK Thumbprint) from CAT token's "cnf" claim
 - * Verify DPoP JWT signature using embedded public key
 - * Confirm that SHA-256 hash of DPoP public key matches "jkt" value
 - * Check proof freshness within "catdpop" window settings
 - * Process replay protection based on "jti" settings

- * Validate Authorization Context ("actx") according to [DPOP-PROOF]
- * Verify "actx.type" is "moqt"
- * Validate "actx.action" matches the requested MOQT action
- * Verify "actx.tns" and "actx.tn" correspond to target resources

5. Action Authorization: Relay validates the specific MOQT action against token scope and namespace/track permissions

6. Response: Relay responds with success or appropriate error information

4. Adding a token to a URL

Any time an application wishes to add a CAT token to a URL or path element, the token SHOULD first be Base64 encoded [BASE64]. The syntax and method of modifying the URL is left to the application to define and is not constrained by this specification.

5. 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.

6. Security Considerations

TODO Add security considerations for DPoP Claims

7. IANA Considerations

IANA will register the following claims in the "CBOR Web Token (CWT) Claims" registry:

	moqt	moqt-reval
Claim Name	moqt	moqt-reval
Claim Description	MOQT Action	MOQT revalidation
JWT Claim Name	N/A	N/A
Claim Key	TBD_MOQT (1+2)	TBD_MOQT (1+2)
Claim Value Type	array	number
Change Controller	IESG	IESG
Specification Document	RFCthis	RFCthis

Table 3

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

8. Normative References

- [BASE64] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [CAT] "CTA 5007-B Common Access Token", April 2025, <<https://shop.cta.tech/products/cta-5007>>.
- [Composite] Lemmons, C., "Composite Token Claims", Work in Progress, Internet-Draft, draft-lemmons-cose-composite-claims-01, 23 July 2025, <<https://datatracker.ietf.org/doc/html/draft-lemmons-cose-composite-claims-01>>.
- [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>>.
- [DPOP-PROOF] Nandakumar, S., "Application-Agnostic Demonstrating Proof-of-Possession", December 2024, <<https://www.ietf.org/archive/id/draft-nandakumar-oauth-dpop-proof-00.txt>>.

[MoQTransport]

Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell,
"Media over QUIC Transport", Work in Progress, Internet-
Draft, draft-ietf-moq-transport-14, 2 September 2025,
<<https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-14>>.

[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>>.

Acknowledgments

The IETF moq workgroup

Authors' Addresses

Will Law
Akamai
Email: wilaw@akamai.com

Chris Lemmons
Comcast
Email: Chris_Lemmons@comcast.com

Gwendal Simon
Synamedia
Email: gsimon@synamedia.com

Suhas Nandakumar
Cisco
Email: snandaku@cisco.com