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

W. Law
Akamai
C. Lemmons
Comcast
G. Simon
Synamedia
S. Nandakumar
Cisco
22 July 2025

Authentication scheme for MOQT using Common Access Tokens
draft-law-moq-cat4moqt-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://wilaw.github.io/CAT-4-MOQT/draft-law-moq-cat4moqt.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-law-moq-cat4moqt/.

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/wilaw/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/.

Copyright Notice

Table of Contents

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 contrains 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 contraints on the actions they can take once connected. The token may be signed to make it tamper-resistent.

* 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 conneciton 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 {draft-lemmons-composite-claims} 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: 'b
ob'}]],
            /exp/ 4: 1750000000
        },
        {
            /moqt/ TBD_MOQT: [[/PUBLISH/ 6, { /exact/ 0: 'example.com'}, { /exact/ 0: 'lo
gs/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 instantanous 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-evalution?

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. Authenticating the connection

The connection to a MOQT distribution relay can take place over a WebTransport or native QUIC connection. In both cases, the token is transferred as a query parameter or else embedded in the URI PATH.

3.1. Appending a token as a query parameter

The query parameter name SHOULD be "CAT" (case-sensitive) and the query parameter value SHOULD be the Base64 encoded [BASE64] token. If more than one token is transferred, then the sequential query parameter names "CAT1", "CAT2" .. "CATN" SHOULD be used.

3.2. Embedding a token in a PATH

The token SHOULD span only a single PATH component and the component SHOULD be prefixed with the string "CAT-". If more than one token is transferred, then they SHOULD occupy different components and SHOULD carry sequential prefixes of "CAT1", "CAT2" .. "CATN".

3.3. Usage with WebTransport

With a WebTransport connection, the token can be transferred as a query parameter or as part of the PATH.

Example of a single token in a query arg:

```
https://example.com/
service?CAT=oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg=
```

Example of multiple tokens in query args:

```
https://example.com/
service?CAT1=oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg=
&CAT2=IHNramRoZmtjc2pkaGYgc2pkaCBhaCBzIGFzS0pEIDthbGtqIA==
```

Example of a single token in the PATH

```
https://example.com/service/CAT-
oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg=/
```

Example of multiple tokens in the PATH:

```
https://example.com/service/
CAT1-oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg=/CAT2-IHN
ramRoZmtjc2pkaGYgc2pkaCBhaCBzIGFzS0pEIDthbGtqIA==/
```

3.4. Usage with Native QUIC

With a native QUIC connection, the query components and PATH are transmitted via the "PATH" parameter in the CLIENT_SETUP message.

Example of a single token in a query arg:

```
moqt://203.0.113.0:4443 PATH parameter in the CLIENT_SETUP message
= "service?CAT=oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg="
```

Example of multiple tokens in query args:

```
moqt://203.0.113.0:4443 PATH parameter in the CLIENT_SETUP message
= "service?CAT1=oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg=
&CAT2=IHNramRoZmtjc2pkaGYgc2pkaCBhaCBzIGFzS0pEIDthbGtqIA=="
```

Example of a single token in the PATH

```
moqt://203.0.113.0:4443 PATH parameter in the CLIENT_SETUP message
= "service/CAT-oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg=/"
```

Example of multiple tokens in the PATH:

```
moqt://203.0.113.0:4443 PATH parameter in the CLIENT_SETUP message
= "service/
CAT1-oRkBDqMAoQBlaHR0cHMDoQFoL2NvbnRlbnQIoQBlLm0zdTg=/CAT2-IHN
ramRoZmtjc2pkaGYgc2pkaCBhaCBzIGFzS0pEIDthbGtqIA==/"
```

4. Controlling access to MOQT actions

TODO

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 Security

7. IANA Considerations

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

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

Table 2

[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-A Common Access Token", December 2024, <https://shop.cta.tech/products/cta-5007>.

[MoQTransport] Curley, L., Pugin, K., Nandakumar, S., Vasiliev, V., and I. Swett, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-10, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-10>.

[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