

Workgroup: ACE Working Group  
Internet-Draft:  
draft-ietf-ace-mqtt-tls-profile-17  
Published: 23 March 2022

Intended Status: Standards Track  
Expires: 24 September 2022

Authors: C.S. Sengul                      A.K. Kirby  
          Brunel University      Oxbotica

**Message Queuing Telemetry Transport (MQTT)-TLS profile of  
Authentication and Authorization for Constrained Environments (ACE)  
Framework**

## **Abstract**

This document specifies a profile for the ACE (Authentication and Authorization for Constrained Environments) framework to enable authorization in a Message Queuing Telemetry Transport (MQTT)-based publish-subscribe messaging system. Proof-of-possession keys, bound to OAuth2.0 access tokens, are used to authenticate and authorize MQTT Clients. The protocol relies on TLS for confidentiality and MQTT server (Broker) authentication.

## **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 24 September 2022.

## **Copyright Notice**

Copyright (c) 2022 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](#)
  - [1.1. Requirements Language](#)
  - [1.2. ACE-Related Terminology](#)
  - [1.3. MQTT-Related Terminology](#)
- [2. Authorizing Connection Requests](#)
  - [2.1. Client Token Request to the Authorization Server \(AS\)](#)
  - [2.2. Client Connection Request to the Broker \(C\)](#)
    - [2.2.1. Overview of Client-RS Authentication Methods over TLS and MQTT](#)
    - [2.2.2. authz-info: The Authorization Information Topic](#)
    - [2.2.3. Client Authentication over TLS](#)
      - [2.2.3.1. Raw Public Key Mode](#)
      - [2.2.3.2. Pre-Shared Key Mode](#)
    - [2.2.4. Client Authentication over MQTT](#)
      - [2.2.4.1. Transporting the Access Token Inside the MQTT CONNECT](#)
      - [2.2.4.2. Authentication Using AUTH Property](#)
    - [2.2.5. Broker Token Validation](#)
  - [2.3. Token Scope and Authorization](#)
  - [2.4. Broker Response to Client Connection Request](#)
    - [2.4.1. Unauthorized Request and the Optional Authorization Server Discovery](#)
    - [2.4.2. Authorization Success](#)
- [3. Authorizing PUBLISH and SUBSCRIBE Packets](#)
  - [3.1. PUBLISH Packets from the Publisher Client to the Broker](#)
  - [3.2. PUBLISH Packets from the Broker to the Subscriber Clients](#)
  - [3.3. Authorizing SUBSCRIBE Packets](#)
- [4. Token Expiration, Update, and Reauthentication](#)
- [5. Handling Disconnections and Retained Messages](#)
- [6. Reduced Protocol Interactions for MQTT v3.1.1](#)
  - [6.1. Token Transport](#)
  - [6.2. Handling Authorization Errors](#)
- [7. IANA Considerations](#)
  - [7.1. TLS Exporter Label Registration](#)
  - [7.2. Media Type Registration](#)
  - [7.3. ACE OAuth Profile Registration](#)
  - [7.4. AIF](#)
- [8. Security Considerations](#)
- [9. Privacy Considerations](#)
- [10. References](#)
  - [10.1. Normative References](#)
  - [10.2. Informative References](#)

[Appendix A. Checklist for profile requirements](#)

[Appendix B. Document Updates](#)

[Acknowledgments](#)

[Authors' Addresses](#)

## 1. Introduction

This document specifies a profile for the ACE framework [[I-D.ietf-ace-oauth-authz](#)]. In this profile, Clients and Servers (Brokers) use MQTT to exchange Application Messages. The protocol relies on TLS for communication security between entities. The MQTT protocol interactions are described based on the [MQTT v5.0 - the OASIS Standard](#) [[MQTT-OASIS-Standard-v5](#)]. Since it is expected that MQTT deployments will continue to support MQTT v3.1.1 Clients, this document also describes a reduced set of protocol interactions for [MQTT v3.1.1 - the OASIS Standard](#) [[MQTT-OASIS-Standard-v3.1.1](#)]. However, MQTT v5.0 is the RECOMMENDED version as it works more naturally with ACE-style authentication and authorization.

MQTT is a publish-subscribe protocol, and after connecting to the MQTT Server (Broker), a Client can publish and subscribe to multiple topics. The Broker, which acts as the Resource Server (RS), is responsible for distributing messages published by the publishers to their subscribers. In the rest of the document, the terms "RS", "MQTT Server" and "Broker" are used interchangeably.

Messages are published under a Topic Name, and subscribers subscribe to the Topic Names to receive the corresponding messages. The Broker uses the Topic Name in a published message to determine which subscribers to relay the messages to. In this document, topics, more specifically, Topic Names, are treated as resources. The Clients are assumed to have identified the publish/subscribe topics of interest out-of-band (topic discovery is not a feature of the MQTT protocol). A Resource Owner can pre-configure policies at the Authorization Server (AS) that give Clients publish or subscribe permissions to different topics.

Clients prove their permission to publish and subscribe to topics hosted on an MQTT Broker using an access token, bound to a proof-of-possession (PoP) key. This document describes how to authorize the following exchanges between the Clients and the Broker.

- \*Connection requests from the Clients to the Broker

- \*Publish requests from the Clients to the Broker and from the Broker to the Clients

- \*Subscribe requests from the Clients to the Broker

Clients use the MQTT PUBLISH packet to publish to a topic. The mechanisms specified in this document do not protect the payload of the PUBLISH packet from the Broker. Hence, the payload is not signed or encrypted specifically for the subscribers. This functionality may be implemented using the proposal outlined in the [ACE Pub-Sub Profile](#) [I-D.ietf-ace-pubsub-profile].

To provide communication confidentiality and Broker authentication to the MQTT Clients, TLS is used, and TLS 1.3 [RFC8446] is RECOMMENDED. This document makes the same assumptions as Section 4 of the [ACE framework](#) [I-D.ietf-ace-oauth-authz] regarding Client and RS registration with the AS and setting up the keying material. While the Client-Broker exchanges are only over MQTT, the required Client-AS and RS-AS interactions are described for HTTPS-based communication [I-D.ietf-httpbis-semantics], using "application/ace+json" content type, and unless otherwise specified, using JSON encoding. The token MAY be an opaque reference to authorization information or JSON Web Token (JWT) [RFC7519]. For JWTs, this document follows [RFC7800] for PoP semantics for JWTs, and the mechanisms for providing and verifying PoP are detailed in [Section 2.2](#). The Client-AS and RS-AS exchanges MAY also use protocols other than HTTP, e.g., Constrained Application Protocol (CoAP) [RFC7252] or MQTT. It is recommended that TLS is used to secure these communication channels between Client-AS and RS-AS. To reduce the protocol memory and bandwidth requirements, implementations MAY also use "application/ace+cbor" content type, and CBOR encoding [RFC8949], and CBOR Web Token (CWT) [RFC8392] and associated PoP semantics. For more information, see [Proof-of-Possession Key Semantics for CBOR Web Tokens \(CWTs\)](#) [RFC8747]. A JWT token uses JOSE, while a CWT token uses COSE [RFC8152] for security protection.

### 1.1. Requirements Language

The keywords "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.

### 1.2. ACE-Related Terminology

Certain security-related terms such as "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify" are taken from [RFC4949].

The terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749] such as "Client" (C), "Resource Server" (RS) and "Authorization Server" (AS).

The term "resource" is used to refer to an MQTT Topic Name, which is defined in [Section 1.3](#). Hence, the "Resource Owner" is any entity that can authoritatively speak for the topic. This document also defines a Client Authorization Server for Clients that are not able to support HTTP.

#### **Client Authorization Server (CAS)**

An entity that prepares and endorses authentication and authorization data for a Client, and communicates to the AS using HTTPS.

### **1.3. MQTT-Related Terminology**

The document describes message exchanges as MQTT protocol interactions. The Clients are MQTT Clients, which connect to the Broker to publish and subscribe to Application Messages, labelled with their topics. For additional information, please refer to the [MQTT v5.0 - the OASIS Standard \[MQTT-OASIS-Standard-v5\]](#) or the [MQTT v3.1.1 - the OASIS Standard \[MQTT-OASIS-Standard-v3.1.1\]](#).

#### **Broker**

The Server in MQTT. It acts as an intermediary between the Clients that publish Application Messages and the Clients that made Subscriptions. The Broker acts as the Resource Server for the Clients.

#### **Client**

A device or program that uses MQTT.

#### **Network Connection**

A construct provided by the underlying transport protocol that is being used by MQTT. It connects the Client to the Server. It

provides the means to send an ordered, lossless, stream of bytes in both directions. This document uses TLS as transport protocol.

### **Session**

A stateful interaction between a Client and a Broker. Some Sessions last only as long as the Network Connection; others can span multiple Network Connections.

### **Application Message**

The data carried by the MQTT protocol. The data has an associated Quality-of-Service (QoS) level and Topic Name.

### **MQTT Control Packet**

The MQTT protocol operates by exchanging a series of MQTT Control packets. Each packet is composed of a Fixed Header, a Variable Header (depending on the control packet type), and a Payload.

### **UTF-8 encoded string**

A string prefixed with a two-byte length field that gives the number of bytes in a UTF-8 encoded string itself. Unless stated otherwise, all UTF-8 encoded strings can have any length in the range 0 to 65535 bytes.

### **Binary Data**

Binary Data is represented by a two-byte length field which indicates the number of data bytes, followed by that number of bytes. Thus, the length of Binary Data is limited to the range of 0 to 65535 Bytes.

### **Variable Byte Integer**

Variable Byte Integer is encoded using an encoding scheme that uses a single byte for values up to 127. For larger values, the least significant seven bits of each byte encode the data, and the most significant bit is used to indicate whether there are bytes following in the representation. Thus, each byte encodes 128 values and a "continuation bit". The maximum number of bytes in the Variable Byte Integer field is four.

### **QoS level**

The level of assurance for the delivery of an Application Message. The QoS level can be 0-2, where 0 indicates "At most once delivery", 1 "At least once delivery", and 2 "Exactly once delivery".

### **Property**

The last field of the Variable Header is a set of properties for several MQTT control packets (e.g. CONNECT, CONNACK). A Property consists of an Identifier that defines its usage and data type, followed by a value. The Identifier is encoded as a Variable Byte

Integer. For example, the "Authentication Data" property uses the Identifier 22.

**Topic Name**

The label attached to an Application Message, which is matched to a Subscription.

**Subscription**

A Subscription comprises a Topic Filter and a maximum QoS. A Subscription is associated with a single session.

**Topic Filter**

An expression that indicates interest in one or more Topic Names. Topic Filters may include wildcards.

MQTT sends various control packets across a Network Connection. The following is not an exhaustive list, and the control packets that are not relevant for authorization are not explained. These include, for instance, the PUBREL and PUBCOMP packets used in the 4-step handshake required for QoS level 2.

**CONNECT**

Client request to connect to the Broker. This is the first packet sent by a Client.

**CONNACK**

The Broker connection acknowledgment. CONNACK packets contain return codes indicating either a success or an error state in response to a Client's CONNECT packet.

**AUTH**

Authentication Exchange. An AUTH control packet is sent from the Client to the Broker or from the Broker to the Client as part of an extended authentication exchange. AUTH Properties include Authentication Method and Authentication Data. The Authentication Method is set in the CONNECT packet, and consequent AUTH packets follow the same Authentication Method. The contents of the Authentication Data are defined by the Authentication Method.

**PUBLISH**

Publish request sent from a publishing Client to the Broker, or from the Broker to a subscribing Client.

**PUBACK**

Response to a PUBLISH request with QoS level 1. A PUBACK can be sent from the Broker to a Client or from a Client to the Broker.

**PUBREC**

Response to PUBLISH request with QoS level 2. PUBREC can be sent from the Broker to a Client or from a Client to the Broker.

**SUBSCRIBE**

Subscribe request sent from a Client.

**SUBACK**

Subscribe acknowledgment from the Broker to the Client.

**PINGREQ**

A ping request sent from a Client to the Broker. It signals to the Broker that the Client is alive and is used to confirm that the Broker is also alive. The "Keep Alive" period is set in the CONNECT packet.

**PINGRESP**

Response sent by the Broker to the Client in response to PINGREQ. It indicates the Broker is alive.

**DISCONNECT**

The DISCONNECT packet is the final MQTT Control Packet sent from the Client or the Broker. It indicates the reason why the Network Connection is being closed. If the Network Connection is closed without the Client first sending a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) and the Connection has a Will Message, the Will Message is published.

**Will**

If the Network Connection is not closed normally, the Broker sends a last Will message for the Client if the Client provided one in its CONNECT packet. Situations in which the Will Message is published include, but are not limited to:

- \*An I/O error or network failure detected by the Broker.

- \*The Client fails to communicate within the Keep Alive period.

- \*The Client closes the Network Connection without first sending a DISCONNECT packet with a Reason Code 0x00 (Normal disconnection).

- \*The Broker closes the Network Connection without first receiving a DISCONNECT packet with a Reason Code 0x00 (Normal disconnection).

If the Will Flag is set in the CONNECT flags, then the payload of the CONNECT packet includes information about the Will. The information consists of the Will Properties, Will Topic, and Will Payload fields.

## 2. Authorizing Connection Requests

This section specifies how Client connections are authorized by the AS and verified by the MQTT Broker. [Figure 1](#) shows the basic protocol flow during connection setup. The token request and response use the token endpoint at the AS, specified for HTTP-based interactions in Section 5.8 of the [ACE framework](#) [[I-D.ietf-ace-oauth-authz](#)]. Steps (D) and (E) are optional and use the introspection endpoint specified in Section 5.9 of the ACE framework. The discussion in this document assumes that the Client and the Broker use HTTPS to communicate with the AS via these endpoints. The Client and the Broker use MQTT to communicate between them. The C-AS and Broker-AS communication MAY be implemented using protocols other than HTTPS, e.g. CoAP or MQTT. Whatever protocol is used for C-AS and Broker-AS communications MUST provide mutual authentication, confidentiality protection, and integrity protection.

If the Client is resource-constrained or does not support HTTPS, a separate Client Authorization Server may carry out the token request on behalf of the Client (Figure 1 (A) and (B)), and later, onboard the Client with the token. The interactions between a Client and its Client Authorization Server for token onboarding and support for MQTT-based token requests at the AS are out of the scope of this document.

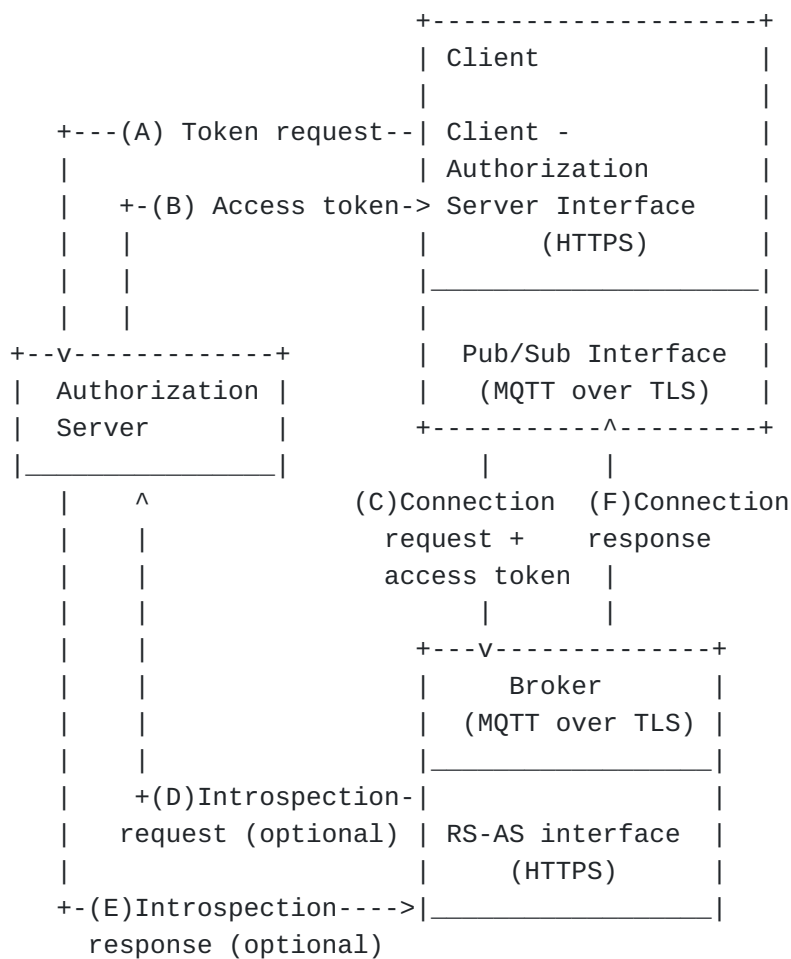


Figure 1: Connection Setup

### 2.1. Client Token Request to the Authorization Server (AS)

The first step in the protocol flow (Figure 1 (A)) is the token acquisition by the Client from the AS. The Client and the AS MUST perform mutual authentication. The Client requests an access token from the AS as described in Section 5.8.1 of the [ACE framework \[I-D.ietf-ace-oauth-authz\]](#). The document follows the procedures defined in Section 3.2.1 of the [DTLS profile \[I-D.ietf-ace-dtls-authorize\]](#) for RPK (Raw Public Keys [[RFC7250](#)]), and in Section 3.3.1 of the same document for PSK (Pre-Shared Keys). However, the content type of the request is set to "application/ace+json", and the AS uses JSON in the payload of its responses to the Client and the RS. As explained earlier, implementations MAY also use "application/ace+cbor" content type.

On receipt of the token request, the AS verifies the request. If the AS successfully verifies the access token request and authorizes the Client for the indicated audience (i.e., RS) and scopes (i.e., publish/subscribe permissions over topics as described in [Section 2.3](#)), the AS issues an access token (Figure 1 (B)).

The response includes the parameters described in Section 5.8.2 of the [ACE framework \[I-D.ietf-ace-oauth-authz\]](#). For RPK, the parameters are as described in Section 3.2.1 of the [DTLS profile \[I-D.ietf-ace-dtls-authorize\]](#). For PSK, the document follows Section 3.3.1 of the [DTLS profile \[I-D.ietf-ace-dtls-authorize\]](#). In both cases, if the response contains an "ace\_profile" parameter, this parameter is set to "mqtt\_tls". The returned token is a Proof-of-Possession (PoP) token by default.

This document follows [\[RFC7800\]](#) for PoP semantics for JWTs (CWTs MAY also be used). The AS includes a "cnf" (confirmation) parameter in the PoP token, to declare that the Client possesses a particular key and RS can cryptographically confirm that the Client has possession of that key, as described in [\[I-D.ietf-ace-oauth-params\]](#).

Note that the contents of the web tokens (including the "cnf" parameter) are to be consumed by the RS and not the Client (the Client obtains the key information in a different manner). The RPK case is handled as described in Section 3.2.1 of the [DTLS profile \[I-D.ietf-ace-dtls-authorize\]](#). For the PSK case, the referenced procedures apply, with the following exceptions to accommodate JWT and JOSE use. In this case, the AS adds a "cnf" parameter to the access information carrying a [JWK \(JSON Web Key\) \[RFC7517\]](#) object that contains either the symmetric key itself or a key identifier that can be used by the RS to determine the secret key it shares with the Client. The JWT is created as explained in Section 7 of [\[RFC7519\]](#), and the JWT MUST include [JWE \[RFC7516\]](#). If CWT/COSE is used this information MUST be inside the "COSE\_Key" object, and MUST be encrypted using a "COSE\_Encrypt0" structure.

The AS returns error responses for JSON-based interactions following Section 5.2 of [\[RFC6749\]](#). When CBOR is used, the interactions MUST implement Section 5.8.3 of the [ACE framework \[I-D.ietf-ace-oauth-authz\]](#).

## **2.2. Client Connection Request to the Broker (C)**

### **2.2.1. Overview of Client-RS Authentication Methods over TLS and MQTT**

Unless the Client publishes and subscribes to only public topics, the Client and the Broker MUST perform mutual authentication. The Client MUST authenticate to the Broker either over MQTT or TLS before performing any other action. For MQTT, the options are "None" and "ace". For TLS, the options are "Anon" for an anonymous client, and "Known(RPK/PSK)" for RPK and PSK, respectively. The "None" and "Anon" options do not provide client authentication but can be used either during authentication or in combination with authentication at the other layer. When the Client uses TLS:Anon,MQTT:None, the Client can only publish or subscribe to public topics. Thus, the

client authentication procedures involve the following possible combinations:

\*TLS:Anon,MQTT:None: This option is used only for the topics that do not require authorization, including the "authz-info" topic. Publishing to the "authz-info" topic is described in [Section 2.2.2](#).

\*TLS:Anon,MQTT:ace: The token is transported inside the CONNECT packet and MUST be validated using one of the methods described in [Section 2.2.2](#). This option also supports a tokenless connection request for AS discovery. As per the [ACE framework \[I-D.ietf-ace-oauth-authz\]](#), a separate step is needed to determine whether the discovered AS URI is authorized to act as an AS.

\*TLS:Known(RPK/PSK),MQTT:none: This specification supports client authentication with TLS with RPK and PSK following the procedures described in [DTLS profile \[I-D.ietf-ace-dtls-authorize\]](#). For the RPK, the Client MUST have published the token to the "authz-info" topic. For the PSK, the token MAY be published to the "authz-info" topic, or MAY be, alternatively, provided as a "PSK identity" (e.g. an "identity" in the "identities" field in the Client's "pre\_shared\_key" extension in TLS 1.3).

\*TLS:Known(RPK/PSK),MQTT:ace: This option SHOULD NOT be chosen as the token transported in the CONNECT overwrites any permissions passed during the TLS authentication.

It is RECOMMENDED that the Client implements TLS:Anon,MQTT:ace as the first choice when working with protected topics. However, MQTT v3.1.1 Clients that do not prefer to overload username and password fields for ACE (as described in [Section 6](#)) MAY implement TLS:Known(RPK/PSK),MQTT:none, and consequently TLS:Anon,MQTT:None to submit their token to "authz-info".

The Broker MUST support TLS:Anon,MQTT:ace. To support Clients with different capabilities, the Broker MAY provide multiple client authentication options, e.g. support TLS:Known(RPK),MQTT:none and TLS:Anon,MQTT:None, to enable RPK-based client authentication.

The Client MUST authenticate the Broker during the TLS handshake. If the Client authentication uses TLS:Known(RPK/PSK), then the Broker is authenticated using the respective method. Otherwise, to authenticate the Broker, the Client MUST validate a public key from an X.509 certificate or an RPK from the Broker against the "rs\_cnf" parameter in the token response, which contains information about the public key used by the RS to authenticate if the token type is "pop" and asymmetric keys are used as defined in [\[I-D.ietf-ace-oauth-params\]](#). The AS MAY include the thumbprint of the RS's X.509

certificate in the "rs\_cnf" (thumbprint as defined in [[I-D.ietf-cose-x509](#)]). In this case, the Client MUST validate the RS certificate against this thumbprint.

### **2.2.2. authz-info: The Authorization Information Topic**

In the cases when the Client must transport the token to the Broker first, the Client connects to the Broker to publish its token to the "authz-info" topic. The "authz-info" topic MUST be publish-only (i.e., the Clients are not allowed to subscribe to it). "authz-info" is not protected, and hence, the Client uses the TLS:Anon, MQTT:None option over a TLS connection. After publishing the token, the Client disconnects from the Broker and is expected to reconnect using client authentication over TLS (i.e., TLS:Known(RPK/PSK), MQTT:none).

The Broker stores and indexes all tokens received to the "authz-info" topic in its key store (similar to the [DTLS profile for ACE](#) [[I-D.ietf-ace-dtls-authorize](#)]). This profile follows the recommendation of Section 5.10.1 of the [ACE framework](#) [[I-D.ietf-ace-oauth-authz](#)] and expects that the Broker stores only one token per PoP key, and any other token linked to the same key overwrites an existing token.

The Broker MUST verify the validity of the token (i.e., through local validation or introspection, if the token is a reference) as described in [Section 2.2.5](#). If the token is not valid, the Broker MUST discard the token.

Depending on the QoS level of the PUBLISH packet, the Broker returns the error response as a PUBACK, PUBREC, or DISCONNECT packet. If the QoS level is equal to 0, and the token is not valid, or the claims cannot be obtained in the case of an introspected token, the Broker MUST send a DISCONNECT packet with the reason code 0x87 (Not authorized). If the PUBLISH payload does not parse to a token, the Broker MUST send a DISCONNECT with the reason code 0x99 (Payload format invalid).

If the QoS level of the PUBLISH packet is greater than or equal to 1, and the token is not valid, or the claims cannot be obtained in the case of an introspected token, the Broker MUST send the reason code 0x87 (Not authorized) in the PUBACK or PUBREC. If the PUBLISH payload does not parse to a token, the PUBACK/PUBREC reason code is 0x99 (Payload format invalid).

It must be noted that when the Broker sends the "Not authorized" response, this corresponds to the token being not valid, and not that the actual PUBLISH packet was not authorized. Given that the "authz-info" is a public topic, this response is not expected to cause confusion.

### 2.2.3. Client Authentication over TLS

This document supports TLS with Raw Public Keys (RPK) [[RFC7250](#)] and with Pre-Shared Keys (PSK). The TLS session setup follows the [DTLS profile for ACE](#) [[I-D.ietf-ace-dtls-authorize](#)], as the profile applies to TLS equally well [[I-D.ietf-ace-extend-dtls-authorize](#)]. When there are exceptions to the DTLS profile, these are explicitly stated in the document. If TLS 1.2 is used, [[RFC7925](#)] describes how TLS can be used for constrained devices, alongside recommended cipher suites. Additionally, TLS 1.2 implementations MUST use the "Extended Main Secret" extension (terminology adopted from [[I-D.ietf-tls-rfc8446bis](#)]) to incorporate the handshake transcript into the main secret [[RFC7627](#)]. TLS implementations SHOULD use the SNI (Server Name Indication) [[RFC6066](#)] and APLN (Application-Layer Protocol Negotiation) [[RFC7301](#)] extensions so the TLS handshake authenticates as much of the protocol context as possible.

#### 2.2.3.1. Raw Public Key Mode

This document follows the procedures defined in Section 3.2.2 of the [DTLS profile for ACE](#) [[I-D.ietf-ace-dtls-authorize](#)] with the following exceptions. The Client MUST upload the access token to the Broker using the method specified in [Section 2.2.2](#) before initiating the handshake.

#### 2.2.3.2. Pre-Shared Key Mode

This document follows the procedures defined in Section 3.3.2 of [DTLS profile for ACE](#) [[I-D.ietf-ace-dtls-authorize](#)] with the following exceptions.

To use TLS 1.3 with pre-shared keys, the Client utilizes the PSK key extension specified in [[RFC8446](#)] using the key conveyed in the "cnf" parameter of the AS response. The same key is bound to the access token in the "cnf" claim. The Client can upload the token as specified in [Section 2.2.2](#) before initiating the handshake. When using a previously uploaded token, the Client MUST indicate during the handshake which previously uploaded access token it intends to use. To do so, it MUST create a "COSE\_Key" or "JWK" structure with the "kid" that was conveyed in the "rs\_cnf" claim in the token response from the AS and the key type "symmetric". This structure is then included as the only element in the "cnf" structure and the encoded value of that "cnf" structure used as a PSK identity in TLS. As an alternative to the access token upload, the Client can provide the most recent access token, JWT or CWT, as a PSK identity.

In contrast to [DTLS profile for ACE](#) [[I-D.ietf-ace-dtls-authorize](#)], a Client MAY omit support for the cipher suites TLS\_PSK\_WITH\_AES\_128\_CCM\_8 and TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CCM\_8.

For TLS 1.2, however, a client MUST support TLS\_ECDHE\_PSK\_WITH\_AES\_128\_GCM\_SHA256 for PSK ([\[RFC8442\]](#)) and TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 for RPK ([\[RFC8422\]](#)), as recommended in [\[RFC7525\]](#) (and adjusted to be a PSK cipher suite as appropriate).

#### **2.2.4. Client Authentication over MQTT**

##### **2.2.4.1. Transporting the Access Token Inside the MQTT CONNECT**

This section describes how the Client transports the token to the Broker inside the CONNECT packet. If this method is used, the Client TLS connection is expected to be anonymous, and the Broker is authenticated during the TLS connection setup. The approach described in this section is similar to an earlier proposal by Fremantle, et al. [\[fremantle14\]](#).

After sending the CONNECT, the Client MUST wait to receive the CONNACK from the Broker. The only packets it is allowed to send are DISCONNECT or AUTH that is in response to the Broker AUTH. Similarly, except for a DISCONNECT and AUTH response from the Client, the Broker MUST NOT process any packets before sending a CONNACK.

[Figure 2](#) shows the structure of the MQTT CONNECT packet used in MQTT v5.0. A CONNECT packet is composed of a fixed header, a variable header, and a payload. The fixed header contains the Control Packet Type (CPT), Reserved, and Remaining Length fields. Remaining Length is a Variable Byte Integer that represents the number of bytes remaining within the current Control Packet, including data in the Variable Header and the Payload. The Variable Header contains the Protocol Name, Protocol Level, Connect Flags, Keep Alive, and Properties fields. The Connect Flags in the variable header specify the properties of the MQTT session. It also indicates the presence or absence of some fields in the Payload. The payload contains one or more encoded fields, namely a unique Client Identifier for the Client, a Will Topic, Will Payload, User Name, and Password. All but the Client Identifier can be omitted depending on the flags in the Variable Header. The Client Identifier identifies the Client to the Broker, and therefore, is unique for each Client. It must be noted that the Client Identifier is an unauthenticated identifier used within the MQTT protocol and so is not bound to the access token.

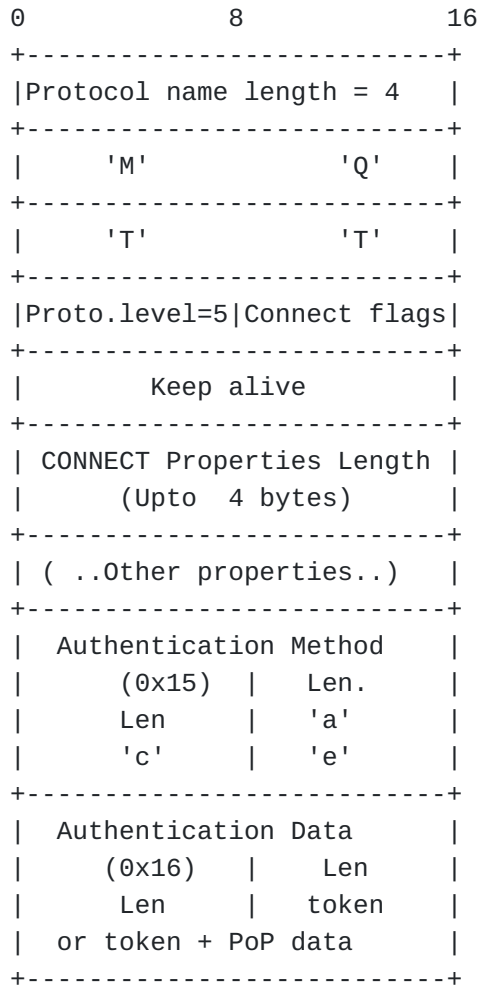


Figure 2: MQTT v5 CONNECT Variable Header with Authentication Method Property for ACE

The CONNECT flags are Username, Password, Will retain, Will QoS, Will Flag, Clean Start, and Reserved. [Figure 3](#) shows how the flags MUST be set to use AUTH packets for authentication and authorization, i.e., the username and password flags MUST be set to 0. An MQTT v5.0 Broker MAY also support token transport using Username and Password to provide a security option for MQTT v3.1.1 Clients, as described in [Section 6](#).

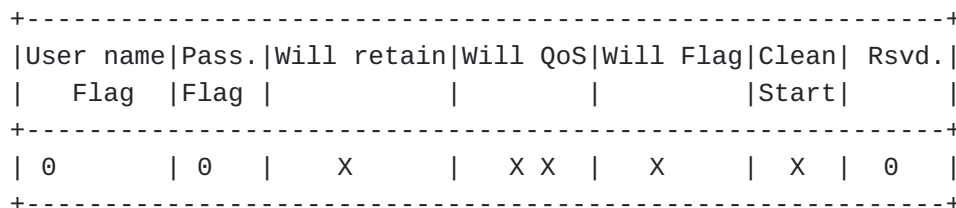


Figure 3: CONNECT Flags for AUTH

The Will Flag indicates that a Will message needs to be sent. The Client MAY set the Will Flag as desired (marked as "X" in [Figure 3](#)). If the Will Flag is set to 1, the Broker MUST check that the token allows the publication of the Will message (i.e., the Will Topic filter is in the scope array). The check is performed against the token scope described in [Section 2.3](#). If the Will authorization fails, the connection is refused as described in [Section 2.4.1](#). If the Broker accepts the connection request, the Broker stores the Will message and publishes it when the Network Connection is closed according to Will QoS, and Will retain parameters and MQTT Will management rules. To avoid publishing the Will Messages in the case of temporary network disconnections, the Client specifies a Will Delay Interval in the Will Properties. [Section 5](#) explains how the Broker deals with the retained messages in further detail.

In MQTT v5.0, the Client signals a clean session (i.e., that the session does not continue an existing session) by setting the Clean Start Flag to 1 in the CONNECT packet. In this profile, the Client SHOULD always start with a clean session. The Broker MAY also signal that it does not support session continuation by setting Session Expiry Interval to 0 in the CONNACK. If the Broker starts a clean session, the Broker MUST set the Session Present flag to 0 in the CONNACK packet to signal this to the Client.

The Broker MAY support session continuation, e.g., if the Broker requires it for QoS reasons. In this case, if a CONNECT packet is received with Clean Start set to 0 and there is a Session associated with the Client Identifier, the Broker MUST resume communications with the Client based on the state from the existing Session. In its response, the Broker MUST set the Session Present flag to 1 in the CONNACK packet to signal session continuation to the Client. The session state stored by the Client and the Broker is described in [Section 5](#).

When reconnecting to a Broker that supports session continuation, the Client MUST still provide a token, in addition to using the same Client Identifier and setting the Clean Start to 0. The Broker MUST still perform PoP validation on the provided token. If the token matches the stored state, the Broker MAY skip introspecting a token-by-reference and use the stored introspection result. The Broker MUST also verify the Client is authorized to receive or send MQTT packets that are pending transmission. When a Client connects with a long Session Expiry Interval, the Broker may need to maintain the Client's MQTT session state after it disconnects for an extended period. Brokers SHOULD implement administrative policies to limit misuse.

Note that, according to the MQTT standard, the Broker uses the Client Identifier to identify the session state. In the case of a

Client Identifier collision, a Client may take over another Client's session. Given that the Broker **MUST** associate the Client with a valid token, a Client will only send or receive messages to its authorized topics. Therefore, while this issue is not expected to affect security, it may affect QoS (i.e., PUBLISH or QoS messages saved for Client A may be delivered to a Client B). In addition, if this Client Identifier represents a Client already connected to the Broker, the Broker sends a DISCONNECT packet to the existing Client with Reason Code of 0x8E (Session taken over) and closes the connection to the Client.

#### 2.2.4.2. Authentication Using AUTH Property

To use AUTH, the Client **MUST** set the Authentication Method as a property of a CONNECT packet by using the property identifier 21 (0x15). This is followed by a UTF-8 Encoded String containing the name of the Authentication Method, which **MUST** be set to "ace". If the Broker does not support this profile, it sends a CONNACK with a Reason Code of 0x8C (Bad authentication method).

The Authentication Method is followed by the Authentication Data, which has a property identifier 22 (0x16) and is Binary Data. Based on the Authentication Data, the Broker **MUST** support both options below:

- \*Proof-of-Possession using a challenge from the TLS session

- \*Proof-of-Possession via Broker-generated challenge/response

##### 2.2.4.2.1. Proof-of-Possession Using a Challenge from the TLS session

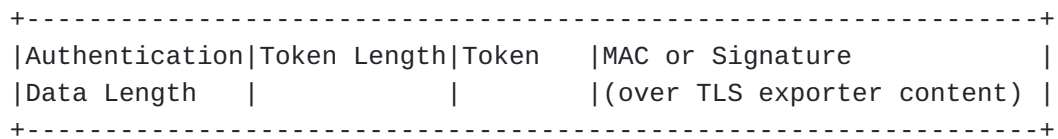


Figure 4: Authentication Data for PoP Based on TLS Exporter Content

For this option, the Authentication Data inside the Client's CONNECT **MUST** contain the two-byte integer token length, the token, and the keyed message digest (MAC) or the Client signature (as shown in [Figure 4](#)). The Proof-of-Possession key in the token is used to calculate the keyed message digest (MAC) or the Client signature based on the content obtained from the TLS exporter ([\[RFC5705\]](#) for TLS 1.2, and Section 7.5 of [\[RFC8446\]](#)) for TLS 1.3. This content is exported from the TLS session using the exporter label "EXPORTER-ACE-MQTT-Sign-Challenge", an empty context, and length of 32 bytes. The token is also validated as described in [Section 2.2.5](#), and the Broker responds with a CONNACK with the appropriate response code.

The Client cannot reauthenticate using this method during the same TLS session (see [Section 4](#)).

#### 2.2.4.2.2. Proof-of-Possession via Broker-generated Challenge/Response

```
+-----+
|Authentication|Token Length|Token      |
|Data Length   |              |           |
+-----+
```

Figure 5: Authentication Data to Initiate PoP Based on Challenge/Response

```
+-----+
|Authentication|RS Nonce    |
|Data Length   |(8 bytes)  |
+-----+
```

Figure 6: Authentication Data for Broker Challenge

For this option, the Broker follows a Broker-generated challenge/response protocol. If the Authentication Data inside the Client's CONNECT contains only the two-byte integer token length and the token (as shown in [Figure 5](#)), the Broker MUST respond with an AUTH packet, with the Authenticate Reason Code set to 0x18 (Continue Authentication). The Broker also uses this method if the Authentication Data does not contain a token, but the Broker has a token stored for the connecting Client.

The Broker continues authentication using an AUTH packet that contains the Authentication Method and the Authentication Data. The Authentication Method MUST be set to "ace", and the Authentication Data MUST NOT be empty and MUST contain an 8-byte RS nonce as a challenge for the Client ([Figure 6](#)).

```
+-----+
|Authentication|Client Nonce |MAC or Signature          |
|Data Length   |(8 bytes)   |(over RS nonce+Client nonce)|
+-----+
```

Figure 7: Authentication Data for Client Challenge Response

The Client responds to this with an AUTH packet with a reason code 0x18 (Continue Authentication). Similarly, the Client packet sets the Authentication Method to "ace". The Authentication Data in the Client's response is formatted as shown in [Figure 7](#) and includes the 8-byte Client nonce, and the signature or MAC computed over the RS nonce concatenated with the Client nonce using PoP key in the token.

Next, the token is validated as described in [Section 2.2.5](#). The success case is illustrated in [Figure 8](#). The Client MAY also re-authenticate using this challenge-response flow, as described in [Section 4](#).

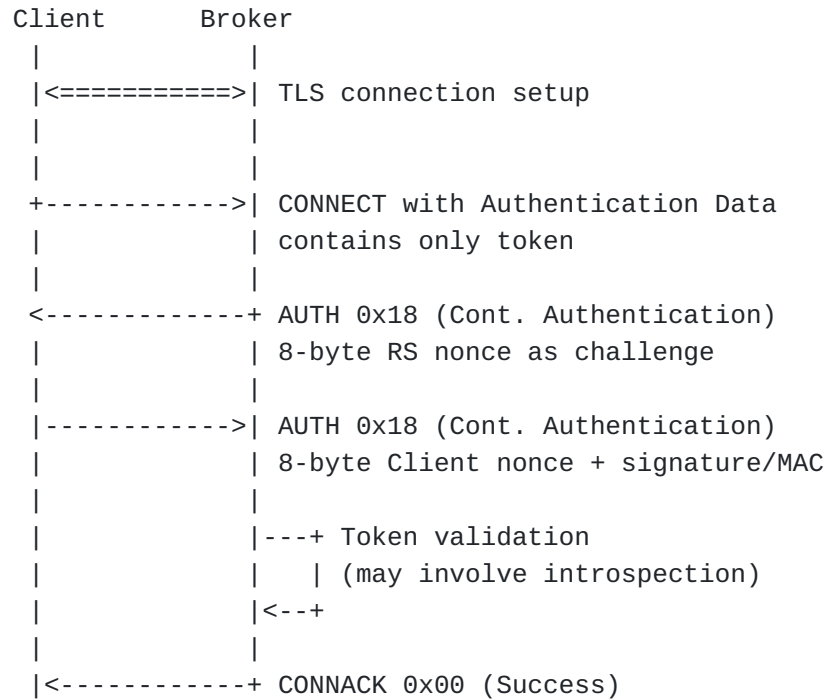


Figure 8: PoP Challenge/Response Flow - Success

### 2.2.5. Broker Token Validation

The Broker MUST verify the validity of the token either locally (e.g., in the case of a self-contained token) or MAY send a request to the introspection endpoint of the AS (as described for HTTP-based interactions in Section 5.9 of the [ACE framework](#) [[I-D.ietf-ace-oauth-authz](#)]). The Broker MUST verify the claims in the access token according to the rules set in Section 5.10.1.1 of the [ACE framework](#) [[I-D.ietf-ace-oauth-authz](#)].

To authenticate the Client, the Broker validates the signature or the MAC, depending on how the PoP protocol is implemented. For self-contained tokens, the Broker MUST process the security protection of the token first, as specified by the respective token format, i.e. a CWT token uses COSE, while a JWT token uses JOSE. For a token-by-reference, the Broker uses the "cnf" structure returned as a result of token introspection as specified in [[RFC7519](#)]. HS256 (HMAC-SHA-256) [[RFC6234](#)] and Ed25519 [[RFC8032](#)] are mandatory to implement for the Broker. The Client MUST implement at least one of them depending on the choice of symmetric or asymmetric validation. Validation of the signature or MAC MUST fail if the signature

algorithm is set to "none", when the key used for the signature algorithm cannot be determined, or the computed and received signature/MAC do not match.

The Broker MUST check if the access token is still valid, if it is the intended destination (i.e., the audience) of the token, and if the token was issued by an authorized authorization server. If the Client is using TLS RPK mode to authenticate to the Broker, the AS constructs the access token so that the Broker can associate the access token with the Client's public key. The "cnf" claim MUST contain either the Client's RPK or, if the key is already known by the Broker (e.g., from previous communication), a reference to it.

### 2.3. Token Scope and Authorization

The scope field contains the publish and subscribe permissions for the Client. Therefore, the token or its introspection result MUST be cached to allow a Client's future PUBLISH and SUBSCRIBE messages. During the CONNECT, if the Will Flag is set to 1, the Broker MUST also authorize the publication of the Will Topic and message using the token's scope field. The Broker uses the scope to match against the Topic Name in a PUBLISH packet (including Will Topic in the CONNECT) or a Topic Filter in a SUBSCRIBE packet.

The scope in the token is a single value. For a JWT, the single scope is base64url encoded string with any padding characters removed, which has an internal structure of a JSON array. For a CWT, this information is represented in CBOR. The internal structure follows the [Authorization Information Format \(AIF\) for ACE \[I-D.ietf-ace-aif\]](#). Using the Concise Data Definition Language (CDDL) [\[RFC8610\]](#), the specific data model for MQTT is:

```
AIF-MQTT = AIF-Generic<mqtt-topic-filter, mqtt-permissions>
AIF-Generic<Toid, Tperm> = [* [Toid, Tperm]]
mqtt-topic-filter = tstr ; as per Section 4.7 of MQTT v5.0
mqtt-permissions = [+permission]
permission = "pub"/"sub"
```

Figure 9: AIF-MQTT data model

Topic filters are implemented according to Section 4.7 of [MQTT v5.0 - the OASIS Standard \[MQTT-OASIS-Standard-v5\]](#). By default, Wildcard Subscriptions are supported, and so, the topic filter may include special wildcard characters. The multi-level wildcard, "#", matches any number of levels within a topic, and the single-level wildcard, "+", matches one topic level. The Broker MAY signal in the CONNACK explicitly whether wildcard subscriptions are supported by returning a CONNACK property "Wildcard Subscription Available". A value of 0

means that Wildcard Subscriptions are not supported. A value of 1 means Wildcard Subscriptions are supported.

Following this model, an example scope may contain:

```
[["topic1",["pub","sub"]],["topic2/#",["pub"]],["+/topic3",["sub"]]]
```

Figure 10: Example scope

This access token gives publish ("pub") and subscribe ("sub") permissions to the "topic1", publish permission to all the subtopics of "topic2", and subscribe permission to all "topic3", skipping one level.

If the scope is empty, the Broker records no permissions for the Client for any topic. In this case, the Client is not able to publish or subscribe to any protected topics. The non-empty scope is used to authorize the Will Topic, if provided, in the CONNECT packet, during connection setup, and if the connection request succeeds, the Topic Names or Topic Filters requested in the future PUBLISH and SUBSCRIBE packets. For the authorization to succeed, the Broker MUST verify that the topic name or filter in question is either an exact match to or a subset of at least one "topic\_filter" in the scope.

## **2.4. Broker Response to Client Connection Request**

Based on the validation result (obtained either via local inspection or using the introspection interface of the AS), the Broker MUST send a CONNACK packet to the Client.

### **2.4.1. Unauthorized Request and the Optional Authorization Server Discovery**

Authentication can fail for the following reasons:

- \*If the Client does not provide a valid token,
- \*the Client omits the Authentication Data field and the Broker has no token stored for the Client,
- \*the token or Authentication data are malformed, or
- \*if the Will flag is set, the authorization checks for the Will topic fails.

The Broker responds with the CONNACK reason code 0x87 (Not Authorized) or any other applicable reason code.

The Broker MAY also trigger AS discovery and include a User Property (identified as property type 38 (0x26)) in the CONNACK for the AS Request Creation Hints. The User Property is a UTF-8 string pair, composed of a name and a value. The name of the User Property MUST be set to "ace\_as\_hint". The value of the user property is a UTF-8 encoded JSON object containing the mandatory "AS" parameter, and the optional parameters "audience", "kid", "cnonce", and "scope" as defined in Section 5.3 of the [ACE framework \[I-D.ietf-ace-oauth-authz\]](#).

#### **2.4.2. Authorization Success**

On success, the reason code of the CONNACK is 0x00 (Success). If the Broker starts a new session, it MUST also set Session Present to 0 in the CONNACK packet to signal a clean session to the Client. Otherwise, it MUST set Session Present to 1.

Having accepted the connection, the Broker MUST be prepared to store the token during the connection and after disconnection for future use. If the token is not self-contained and the Broker uses token introspection, it MAY cache the validation result to authorize the subsequent PUBLISH and SUBSCRIBE packets. PUBLISH and SUBSCRIBE packets, which are sent after a connection setup, do not contain access tokens. If the introspection result is not cached, the Broker needs to introspect the saved token for each request. The Broker SHOULD also use a cache timeout to introspect tokens regularly. The timeout value is application-specific and should be chosen to reduce the risk of using stale introspection responses.

### **3. Authorizing PUBLISH and SUBSCRIBE Packets**

Using the cached token or its introspection result, the Broker uses the scope field to match against the Topic Name in a PUBLISH packet, or a Topic Filter in a SUBSCRIBE packet.

#### **3.1. PUBLISH Packets from the Publisher Client to the Broker**

On receiving the PUBLISH packet, the Broker MUST use the type of packet (i.e., PUBLISH) and the Topic name in the packet header to match against the scope array items in the cached token or its introspection result. Following the example in [Section 2.3](#), the Client sending a PUBLISH for "topic2/a" would be allowed, as the scope array includes the ["topic2/#", "pub"]].

If the Client is allowed to publish to the topic, the Broker publishes the message to all valid subscribers of the topic. In the case of an authorization failure, the Broker MUST return an error if the Client has set the QoS level of the PUBLISH packet to greater than or equal to 1. Depending on the QoS level, the Broker responds with either a PUBACK or PUBREC packet with reason code 0x87 (Not

authorized). On receiving an acknowledgment with 0x87 (Not authorized), the Client MAY reauthenticate by providing a new token as described in [Section 4](#).

For QoS level 0, the Broker sends a DISCONNECT with reason code 0x87 (Not authorized) and closes the Network Connection. Note that the server-side DISCONNECT is a new feature of MQTT v5.0 (in MQTT v3.1.1, the server needs to drop the connection).

For all QoS levels, the Broker MAY return 0x80 Unspecified error if they do not want to leak the topic names to unauthorized clients.

### **3.2. PUBLISH Packets from the Broker to the Subscriber Clients**

To forward PUBLISH packets to the subscribing Clients, the Broker identifies all the subscribers that have valid matching topic subscriptions to the Topic name of the PUBLISH packet (i.e., the tokens are valid, and token scopes allow a subscription to this particular Topic name). The Broker forwards the PUBLISH packet to all the valid subscribers.

The Broker MUST NOT forward messages to unauthorized subscribers. To avoid silently dropping messages, the Broker MUST close the network connection and SHOULD inform the affected subscribers. The only way to inform a client, in this case, would be sending a DISCONNECT packet. Therefore, the Broker SHOULD send a DISCONNECT packet with the reason code 0x87 (Not authorized) before closing the network connection to these clients.

### **3.3. Authorizing SUBSCRIBE Packets**

In MQTT, a SUBSCRIBE packet is sent from a Client to the Broker to create one or more subscriptions to one or more topics. The SUBSCRIBE packet may contain multiple Topic Filters. The Topic Filters may include wildcard characters.

On receiving the SUBSCRIBE packet, the Broker MUST use the type of packet (i.e., SUBSCRIBE) and the Topic Filter in the packet header to match against the scope field of the stored token or introspection result. The Topic Filters MUST be an exact match to or be a subset of at least one of the "topic\_filter" fields in the scope array found in the Client's token. For example, if the Client sends a subscription request for topic "a/b/\*", and has a token that permits "a/\*", this is a valid subscription request, as "a/b/\*" is a subset of "a/\*". (The process is similar to a Broker matching the Topic Name in a PUBLISH packet against the Subscriptions known to the Server.)

As a response to the SUBSCRIBE packet, the Broker issues a SUBACK. For each Topic Filter, the SUBACK packet includes a return code

matching the QoS level for the corresponding Topic Filter. In the case of failure, the return code is 0x87, indicating that the Client is not authorized. The Broker MAY return 0x80 Unspecified error if they do not want to leak the topic names to unauthorized clients. A reason code is returned for each Topic Filter. Therefore, the Client may receive success codes for a subset of its Topic Filters while being unauthorized for the rest.

#### **4. Token Expiration, Update, and Reauthentication**

The Broker MUST check for token expiration whenever a CONNECT, PUBLISH, or SUBSCRIBE is received or sent. The Broker SHOULD check for token expiration on receiving a PINGREQUEST. The Broker MAY also check for token expiration periodically, e.g., every hour. This may allow for early detection of a token expiry.

The token expiration is checked by checking the "exp" claim of a JWT or introspection response or via performing an introspection request with the AS as described in Section 5.9 of the [ACE framework \[I-D.ietf-ace-oauth-authz\]](#). Token expirations may trigger the Broker to send PUBACK, SUBACK and DISCONNECT packets with return code set to "Not authorized". After sending a DISCONNECT, the Network Connection is closed, and no more messages can be sent.

The Client MAY reauthenticate as a response to the PUBACK and SUBACK that signal loss of authorization. The Clients MAY also proactively update their tokens, i.e., before they receive a packet with a "Not authorized" return code. To start reauthentication, the Client MUST send an AUTH packet with the reason code 0x19 (Re-authentication). The Client MUST set the Authentication Method as "ace" and transport the new token in the Authentication Data. If re-authenticating during the current TLS session, the Client MUST NOT use the method described in [Section 2.2.4.2.1](#), Proof-of-Possession using a challenge from the TLS session, to avoid re-using the same challenge value from the TLS-Exporter. Note that this means that servers will either need to record in the session ticket or database entry whether the TLS-Exporter-derived challenge was used, or always deny use of the TLS-Exporter-derived challenge for resumed sessions. In TLS 1.3, the resumed connection would have a new exporter value, but the requirement is phrased this way for simplicity. For re-authentications in the same TLS-session, the Client MUST use the challenge-response PoP as defined in [Section 2.2.4.2.2](#). The Broker accepts reauthentication requests if the Client has already submitted a token (may be expired), for which it performed proof-of-possession. Otherwise, the Broker MUST deny the request. If the reauthentication fails, the Broker MUST send a DISCONNECT with the reason code 0x87 (Not Authorized).

## 5. Handling Disconnections and Retained Messages

In the case of a Client DISCONNECT, if the Session Expiry Interval is set to 0, the Broker doesn't maintain session state but MUST keep the retained messages. If the Broker maintains session state, the state MAY include the token and its introspection result (for reference tokens) in addition to the MQTT session state. The MQTT session state is identified by the Client Identifier and includes the following:

- \*Client subscription state,

- \*messages with QoS levels 1 and 2, and which have not been completely acknowledged or are pending transmission to the Client, and

- \*if the Session is currently not connected, the time at which the Session will end and Session State will be discarded.

The token/introspection state is not part of the MQTT session state, and PoP validation is required for each new connection, regardless of whether MQTT session continuation is used.

The messages to be retained are indicated to the Broker by setting a RETAIN flag in a PUBLISH packet. This way, the publisher signals to the Broker to store the most recent message for the associated topic. Hence, the new subscribers can receive the last sent message from the publisher for that particular topic without waiting for the next PUBLISH packet. The Broker MUST continue publishing the retained messages as long as the associated tokens are valid. In the MQTT standard, if QoS is 0 for the PUBLISH packet, the Broker may discard the retained message any time. For QoS>1, the message expiry interval dictates how long the retained message is kept. However, it is important that the Broker avoids sending messages indefinitely for the Clients that never update their tokens (i.e., the Client connects briefly with a valid token, sends a PUBLISH packet with RETAIN flag set to 1 and QoS>1, disconnects, and never connects again). Therefore, the Broker MUST use the minimum of token expiry and message expiry interval to discard a retained message.

In case of disconnections due to network errors or server disconnection due to a protocol error (which includes authorization errors), the Will message is sent if the Client supplied a Will in the CONNECT packet. The Client's token scope array MUST include the Will Topic. The Will message MUST be published to the Will Topic regardless of whether the corresponding token has expired (as it has been validated and accepted during CONNECT).

## 6. Reduced Protocol Interactions for MQTT v3.1.1

This section describes a reduced set of protocol interactions for the MQTT v3.1.1 Clients. An MQTT v5.0 Broker MAY implement these interactions for the MQTT v3.1.1 Clients; The flows described in this section are NOT RECOMMENDED for use by MQTT v5.0 Clients. Brokers that do not support MQTT v3.1.1 Clients return a CONNACK packet with Reason Code 0x84 (Unsupported Protocol Version) in response to the connection requests.

### 6.1. Token Transport

As in MQTT v5.0, the token MAY either be transported before, by publishing to the "authz-info" topic, or inside the CONNECT packet. If the Client provided the token via the "authz-info" topic and will not update the token in the CONNECT packet, it MUST authenticate over TLS. The Broker SHOULD still be prepared to store the Client access token for future use (regardless of the method of transport).

In MQTT v3.1.1, after the Client has published to the "authz-info" topic, the Broker cannot communicate the result of the token validation because PUBACK reason codes or server-side DISCONNECT packets are not supported. In any case, the subsequent TLS handshake would fail without a valid token, which can prompt the Client to obtain a valid token.

To transport the token to the Broker inside the CONNECT packet, the Client uses the username and password fields. [Figure 11](#) shows the structure of the MQTT CONNECT packet.

0	8	16
+-----+		
Protocol name length = 4		
+-----+		
'M'	'Q'	
+-----+		
'T'	'T'	
+-----+		
Proto.level=5 Connect flags		
+-----+		
Keep alive		
+-----+		
Payload		
Client Identifier		
(UTF-8 encoded string)		
Username as access token		
(UTF-8 encoded string)		
Password for signature/MAC		
(Binary Data)		
+-----+		

Figure 11: MQTT CONNECT Variable Header Using Username and Password for ACE

[Figure 12](#) shows how the MQTT connect flags MUST be set to initiate a connection with the Broker.

+-----+						
User name	Pass.	Will retain	Will QoS	Will Flag	Clean	Rsvd.
flag	flag					
+-----+						
1	1	X	X X	X	X	0
+-----+						

Figure 12: MQTT CONNECT Flags (Rsvd=Reserved)

The Client SHOULD set the Clean flag to 1 to always start a new session. If the Clean flag is set to 0, the Broker MUST resume communications with the Client based on the state from the current Session (as identified by the Client Identifier). If there is no Session associated with the Client Identifier, the Broker MUST create a new session. The Broker MUST set the Session Present flag in the CONNACK packet accordingly, i.e., 0 to indicate a clean session to the Client and 1 to indicate session continuation. The Broker MUST still perform PoP validation on the provided Client token. MQTT v3.1.1 does not use a Session Expiry Interval, and the Client expects that the Broker maintains the session state after it disconnects. However, stored Session state can be discarded as a

result of administrator action or policies (e.g. defining an automated response based on storage capabilities), and Brokers SHOULD implement administrative policies to limit misuse.

The Client MAY set the Will Flag as desired (marked as "X" in [Figure 12](#)). Username and Password flags MUST be set to 1 to ensure that the Payload of the CONNECT packet includes both Username and Password fields. The MQTT Username is a UTF-8 encoded string, and the MQTT Password is Binary Data.

The CONNECT in MQTT v3.1.1 does not have a field to indicate the authentication method. To signal that the Username field contains an ACE token, this field MUST be prefixed with "ace" keyword, i.e., the Username field is a concatenation of 'a', 'c', 'e' and the access token represented as:

```
'U+0061' || 'U+0063' || 'U+0065' || UTF-8(access token)
```

Figure 13: Username in CONNECT

To this end, the access token MUST be base64url encoded, omitting the '=' padding characters [[RFC4648](#)].

The password field MUST be set to the keyed message digest (MAC) or signature associated with the access token for PoP. The Client MUST apply the PoP key on the challenge derived from the TLS session as described in [Section 2.2.4.2.1](#).

## 6.2. Handling Authorization Errors

Error handling is more primitive in MQTT v3.1.1 due to not having appropriate error fields, error codes, and server-side DISCONNECTs. Therefore, the Broker will disconnect on almost any error and may not keep the session state, necessitating that clients make a greater effort to ensure that tokens remain valid and do not attempt to publish to topics that they do not have permissions for. The following lists how the Broker responds to specific errors.

\*CONNECT without a token: The tokenless CONNECT attempt MUST fail. This is because the challenge-response based PoP is not possible for MQTT v3.1.1. It is also not possible to support AS discovery since a CONNACK packet in MQTT v3.1.1 does not include a means to provide additional information to the Client. Therefore, AS discovery needs to take place out-of-band.

\*Client-Broker PUBLISH authorization failure: In the case of a failure, it is not possible to return an error in MQTT v3.1.1. Acknowledgment messages only indicate success. In the case of an authorization error, the Broker MUST ignore the PUBLISH packet

and disconnect the Client. Also, as DISCONNECT packets are only sent from a Client to the Broker, the server disconnection needs to take place below the application layer.

\*SUBSCRIBE authorization failure: In the SUBACK packet, the return code is 0x80 indicating failure for the unauthorized topic(s). Note that, in both MQTT versions, a reason code is returned for each Topic Filter.

\*Broker-Client PUBLISH authorization failure: When the Broker is forwarding PUBLISH packets to the subscribed Clients, it may discover that some of the subscribers are no longer authorized due to expired tokens. These token expirations MUST lead to disconnecting the Client rather than silently dropping messages.

## **7. IANA Considerations**

Note to RFC Editor: Please replace all occurrences of "[this document]" with the RFC number of this specification and delete this paragraph.

### **7.1. TLS Exporter Label Registration**

This document registers "EXPORTER-ACE-MQTT-Sign-Challenge" (introduced in [Section 2.2.4.2.1](#) in this document) in the TLS Exporter Label Registry [[RFC8447](#)].

\*Recommended: No

\*DTLS-OK: No

\*Reference: [This document]

### **7.2. Media Type Registration**

This document registers the "application/ace+json" media type for messages of the protocols defined in this document carrying parameters encoded in JSON.

\*Type name: application

\*Subtype name: ace+json

\*Required parameters: N/A

\*Optional parameters: N/A

\*Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type.

\*Security considerations: Section 8 of [this document]

\*Interoperability considerations: none

\*Published specification: [this document]

\*Applications that use this media type: This media type is intended for authorization server-client and authorization server-resource server communication as part of the ACE framework using JSON encoding as specified in [this document].

\*Fragment identifier considerations: none

\*Additional information:

-Deprecated alias names for this type: none

-Magic number(s): none

-File extension(s): none

-Macintosh file type code(s): none

\*Person & email address to contact for further information: Cigdem Sengul (csengul@acm.org)

\*Intended usage: COMMON

\*Restrictions on usage: none

\*Author: Cigdem Sengul (csengul@acm.org)

\*Change controller: IETF

\*Provisional registration? (standards tree only): no

### **7.3. ACE OAuth Profile Registration**

The following registrations are done for the ACE OAuth Profile Registry following the procedure specified in [[I-D.ietf-ace-oauth-authz](#)].

\*Name: mqtt\_tls

\*Description: Profile for delegating Client authentication and authorization using MQTT for the Client and Broker (RS) interactions, and HTTP for the AS interactions. TLS is used for confidentiality and integrity protection and server authentication. Client authentication can be provided either via

TLS or using in-band PoP validation at the MQTT application layer.

\*CBOR Value: To be assigned by IANA in the (-256, 255) range

\*Reference: [this document]

#### 7.4. AIF

For the media-types application/aif+cbor and application/aif+json defined in Section 5.1 of [[I-D.ietf-ace-aif](#)], IANA is requested to register the following entries for the two media-type parameters Toid and Tperm, in the respective sub-registry defined in Section 5.2 of [[I-D.ietf-ace-aif](#)] within the "MIME Media Type Sub-Parameter" registry group.

For Toid:

\*Name: mqtt-topic-filter

\*Description/Specification: Topic Filter as defined in [Section 2.3](#).

\*Reference: [[This document]] ([Section 2.3](#))

For Tperm:

\*Name: mqtt-permissions

\*Description/Specification: Permissions for MQTT client as defined in [Section 2.3](#). Tperm is an array of one or more text strings that each have a value of either "pub" or "sub".

\*Reference: [[This document]] ([Section 2.3](#))

## 8. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [[I-D.ietf-ace-oauth-authz](#)]. Therefore, the security considerations outlined in [[I-D.ietf-ace-oauth-authz](#)] apply to this work.

In addition, the security considerations outlined in [MQTT v5.0 - the OASIS Standard](#) [[MQTT-OASIS-Standard-v5](#)] and [MQTT v3.1.1 - the OASIS Standard](#) [[MQTT-OASIS-Standard-v3.1.1](#)] apply. Mainly, this document provides an authorization solution for MQTT, the responsibility of which is left to the specific implementation in the MQTT standards. In the following, we comment on a few relevant issues based on the current MQTT specifications.

After the Broker validates an access token and accepts a connection from a client, it caches the token to authorize a Client's publish and subscribe requests in an ongoing session. The Broker does not cache any tokens that cannot be validated. If a Client's permissions get revoked, but the access token has not expired, the Broker may still grant publish/subscribe to revoked topics. If the Broker caches the token introspection responses, then the Broker SHOULD use a reasonable cache timeout to introspect tokens regularly. The timeout value is application-specific and should be chosen to reduce the risk of using stale introspection responses. When permissions change dynamically, it is expected that AS also follows a reasonable expiration strategy for the access tokens.

The Broker may monitor Client behaviour to detect potential security problems, especially those affecting availability. These include repeated token transfer attempts to the public "authz-info" topic, repeated connection attempts, abnormal terminations, and Clients that connect but do not send any data. If the Broker supports the public "authz-info" topic, described in [Section 2.2.2](#), then this may be vulnerable to a DDoS attack, where many Clients use the "authz-info" public topic to transport tokens that are not meant to be used, and which the Broker may need to store until the tokens expire.

For MQTT v5.0, when a Client connects with a long Session Expiry Interval, the Broker may need to maintain the Client's MQTT session state after it disconnects for an extended period. For MQTT v3.1.1, the session state may need to be stored indefinitely, as it does not have a Session Expiry Interval feature. The Broker SHOULD implement administrative policies to limit misuse of the session continuation by the Client.

## 9. Privacy Considerations

The privacy considerations outlined in [[I-D.ietf-ace-oauth-authz](#)] apply to this work.

In MQTT, the Broker is a central trusted party and may forward potentially sensitive information between Clients. The mechanisms defined in this document do not protect the contents of the PUBLISH packet from the Broker, and hence, the content of the PUBLISH packet is not signed or encrypted separately for the subscribers. This functionality may be implemented using the proposal outlined in [the ACE Pub-Sub Profile](#) [[I-D.ietf-ace-pubsub-profile](#)]. However, this solution would still not provide privacy for other fields of the packet, such as Topic Name.

## 10. References

### 10.1. Normative References

[I-D.ietf-ace-aif] Bormann, C., "An Authorization Information Format (AIF) for ACE", Work in Progress, Internet-Draft, draft-ietf-ace-aif-07, 15 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-aif-07.txt>>.

[I-D.ietf-ace-dtls-authorize] Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.txt>>.

[I-D.ietf-ace-extend-dtls-authorize] Bergmann, O., Mattsson, J. P., and G. Selander, "Extension of the CoAP-DTLS Profile for ACE to TLS", Work in Progress, Internet-Draft, draft-ietf-ace-extend-dtls-authorize-02, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-extend-dtls-authorize-02.txt>>.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.

[I-D.ietf-ace-oauth-params] Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-params-16, 7 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-16.txt>>.

[I-D.ietf-cose-x509] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X. 509 certificates", Work in Progress, Internet-Draft, draft-ietf-cose-x509-08, 14 December 2020, <<https://www.ietf.org/internet-drafts/draft-ietf-cose-x509-08.txt>>.

[I-D.ietf-httpbis-semantics] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September

2021, <<https://www.ietf.org/archive/id/draft-ietf-httpbis-semantics-19.txt>>.

[MQTT-OASIS-Standard-v3.1.1] Banks, A., Ed. and R. Gupta, Ed., "OASIS Standard MQTT Version 3.1.1 Plus Errata 01", 2015, <<https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>>.

[MQTT-OASIS-Standard-v5] Banks, A., Ed., Briggs, E., Ed., Borgendale, K., Ed., and R. Gupta, Ed., "OASIS Standard MQTT Version 5.0", 2017, <<https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html>>.

[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/info/rfc2119>>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/

RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.

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

[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/info/rfc7519>>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <<https://www.rfc-editor.org/info/rfc7627>>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/info/rfc7800>>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

[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/info/rfc8174>>.

[RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <<https://www.rfc-editor.org/info/rfc8422>>.

[RFC8442] Mattsson, J. and D. Migault, "ECDHE\_PSK with AES-GCM and AES-CCM Cipher Suites for TLS 1.2 and DTLS 1.2", RFC 8442, DOI 10.17487/RFC8442, September 2018, <<https://www.rfc-editor.org/info/rfc8442>>.

**[RFC8446]**

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

**[RFC8610]**

Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

**[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/info/rfc8747>>.

## **10.2. Informative References**

**[fremantle14]**

Fremantle, P., Aziz, B., Kopecky, J., and P. Scott, "Federated Identity and Access Management for the Internet of Things", research International Workshop on Secure Internet of Things, September 2014, <<https://dx.doi.org/10.1109/SIoT.2014.8>>.

**[I-D.ietf-ace-pubsub-profile]**

Palombini, F. and C. Sengul, "Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-pubsub-profile-04, 29 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-pubsub-profile-04.txt>>.

**[I-D.ietf-tls-rfc8446bis]**

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-04, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-tls-rfc8446bis-04.txt>>.

**[RFC4949]**

Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.

**[RFC7252]**

Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

**[RFC7525]**

Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security

(DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.

[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/info/rfc8392>>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <<https://www.rfc-editor.org/info/rfc8447>>.

[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/info/rfc8949>>.

## Appendix A. Checklist for profile requirements

Based on the requirements on profiles for the ACE framework [[I-D.ietf-ace-oauth-authz](#)], this document fulfills the following:

- \*Optional AS discovery: AS discovery is supported with the MQTT v5.0 described in [Section 2.2](#).
- \*The communication protocol between the Client and Broker (RS): MQTT
- \*The security protocol between the Client and RS: TLS
- \*Client and RS mutual authentication: Several options are possible and described in [Section 2.2.1](#).
- \*Proof-of-possession protocols: Specified in [Section 2.2.4.2](#); both symmetric and asymmetric keys supported.
- \*Content format: For the HTTPS interactions with AS, "application/ace+json".
- \*Unique profile identifier: mqtt\_tls
- \*Token introspection: RS uses HTTPS introspect interface of AS.

\*Token request: Client or its Client AS uses the HTTPS token endpoint of the AS.

\*authz-info endpoint: It MAY be supported using the method described in [Section 2.2.2](#), but is not protected other than by the TLS channel between Client and RS.

\*Token transport: Via "authz-info" topic, or TLS with PSK, provided as a PSK identity, or in MQTT CONNECT packet for both versions of MQTT. The AUTH extensions can also be used for authentication and re-authentication for MQTT v5.0, as described in [Section 2.2](#) and [Section 4](#).

## **Appendix B. Document Updates**

Version 15: Addressing GENART review comments.

Version 11 to 15: Addressing AD review comments.

Version 10 to 11: Clarified the TLS use between RS-AS and Client-AS.

Version 09 to 10: Fixed version issues for references.

Version 08 to 09: Fixed spacing issues and references.

Version 07 to 08:

\*Fixed several nits, typos based on WG reviews.

\*Added missing references.

\*Added the definition for Property defined by MQTT, and Client Authorization Server.

\*Added artwork to show Authorization Data format for various PoP-related message exchange.

\*Removed all MQTT-related must/should/may.

\*Made AS discovery optional.

\*Clarified what the client and server must implement for client authentication; cleaned up TLS 1.3 related language.

Version 06 to 07:

\*Corrected the title.

\*In Section 2.2.3, added the constraint on which packets the Client can send, and the server can process after CONNECT before CONNACK.

\*In Section 2.2.3, clarified that session state is identified by Client Identifier, and listed its content.

\*In Section 2.2.3, clarified the issue of Client Identifier collision, when the Broker supports session continuation.

\*Corrected the buggy scope example in Section 3.1.

#### Version 05 to 06:

\*Replace the originally proposed scope format with AIF model. Defined the AIF-MQTT, gave an example with a JSON array. Added a normative reference to the AIF draft.

\*Clarified client connection after submitting token via "authz-info" topic as TLS:Known(RPK/PSK),MQTT:none.

\*Expanded acronyms on their first use including the ones in the title.

\*Added a definition for "Session".

\*Corrected "CONNACK" definition, which earlier said it's the first packet sent by the Broker.

\*Added a statement that the Broker will disconnect on almost any error and may not keep session state.

\*Clarified that the Broker does not cache tokens that cannot be validated.

#### Version 04 to 05:

\*Reorganised Section 2 such that "Unauthorized Request: Authorization Server Discovery" is presented under Section 2.

\*Fixed Figure 2 to remove the "empty" word.

\*Clarified that MQTT v5.0 Brokers may implement username/password option for transporting the ACE token only for MQTT v.3.1.1 clients. This option is not recommended for MQTT v.5.0 clients.

\*Changed Clean Session requirement both for MQTT v.5.0 and v.3.1.1. The Broker SHOULD NOT, instead of MUST NOT, continue sessions. Clarified expected behaviour if session continuation is

supported. Added to the Security Considerations the potential misuse of session continuation.

- \*Fixed the Authentication Data to include token length for the Challenge/Response PoP.

- \*Added that Authorization Server Discovery is triggered if a token is not valid and not only missing.

- \*Clarified that the Broker should not accept any other packets from Client after CONNECT and before sending CONNACK.

- \*Added that client reauthentication is accepted only for the challenge/response PoP.

- \*Added Ed25519 as mandatory to implement.

- \*Fixed typos.

Version 03 to 04:

- \*Linked the terms Broker and MQTT server more at the introduction of the document.

- \*Clarified support for MQTTv3.1.1 and removed phrases that might be considered as MQTTv5 is backwards compatible with MQTTv3.1.1

- \*Corrected the Informative and Normative references.

- \*For AS discovery, clarified the CONNECT message omits the Authentication Data field. Specified the User Property MUST be set to "ace\_as\_hint" for AS Request Creation Hints.

- \*Added that MQTT v5 brokers MAY also implement reduced interactions described for MQTTv3.1.1.

- \*Added to Section 3.1, in case of an authorization failure and QoS level 0, the RS sends a DISCONNECT with reason code 0x87 (Not authorized).

- \*Added a pointer to section 4.7 of MQTTv5 spec for more information on topic names and filters.

- \*Added HS256 and RSA256 are mandatory to implement depending on the choice of symmetric or asymmetric validation.

- \*Added MQTT to the TLS exporter label to make it application specific: 'EXPORTER-ACE-MQTT-Sign-Challenge'.

- \*Added a format for Authentication Data so that length values prefix the token (or client nonce) when Authentication Data contains more than one piece of information.

- \*Clarified clients still connect over TLS (server-side) for the authz-info flow.

#### Version 02 to 03:

- \*Added the option of Broker certificate thumbprint in the 'rs\_cnf' sent to the Client.

- \*Clarified the use of a random nonce from the TLS Exporter for PoP, added to the IANA requirements that the label should be registered.

- \*Added a client nonce, when Challenge/Response Authentication is used between Client and Broker.

- \*Clarified the use of the "authz-info" topic and the error response if token validation fails.

- \*Added clarification on wildcard use in scopes for publish/subscribe permissions

- \*Reorganised sections so that token authorization for publish/subscribe messages are better placed.

#### Version 01 to 02:

- \*Clarified protection of Application Message payload as out of scope, and cited draft-palombini-ace-coap-pubsub-profile for a potential solution

- \*Expanded Client connection authorization to capture different options for Client and Broker authentication over TLS and MQTT

- \*Removed Payload (and specifically Client Identifier) from proof-of-possession in favor of using tls-exporter for a TLS-session based challenge.

- \*Moved token transport via "authz-info" topic from the Appendix to the main text.

- \*Clarified Will scope.

- \*Added MQTT AUTH to terminology.

- \*Typo fixes, and simplification of figures.

Version 00 to 01:

- \*Present the MQTTv5 as the RECOMMENDED version, and MQTT v3.1.1 for backward compatibility.
- \*Clarified Will message.
- \*Improved consistency in the use of terminology and upper/lower case.
- \*Defined Broker and MQTTS.
- \*Clarified HTTPS use for C-AS and RS-AS communication. Removed reference to actors document, and clarified the use of client authorization server.
- \*Clarified the Connect message payload and Client Identifier.
- \*Presented different methods for passing the token and PoP.
- \*Added new figures to explain AUTH packets exchange, updated CONNECT message figure.

## Acknowledgments

The authors would like to thank Ludwig Seitz for his review and his input on the authorization information endpoint; Benjamin Kaduk for his review, insightful comments, and contributions to resolving issues; and Carsten Bormann for his review and revisions to the AIF-MQTT data model. The authors would like to thank Paul Fremantle for the initial discussions on MQTT v5.0 support.

## Authors' Addresses

Cigdem Sengul  
Brunel University  
Dept. of Computer Science  
Uxbridge  
UB8 3PH  
United Kingdom

Email: [csengul@acm.org](mailto:csengul@acm.org)

Anthony Kirby  
Oxbotica  
1a Milford House, Mayfield Road, Summertown  
Oxford  
OX2 7EL  
United Kingdom

Email: [anthony@anthony.org](mailto:anthony@anthony.org)