AMQP Claims-based Security Version 1.0

Committee Specification Draft 01

17 March 2021

This stage:
https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/csd01/amqp-cbs-v1.0-csd01.docx (Authoritative)
https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/csd01/amqp-cbs-v1.0-csd01.html
https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/csd01/amqp-cbs-v1.0-csd01.pdf

Previous stage of Version 1.0:
N/A

Latest stage of Version 1.0:
https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/amqp-cbs-v1.0.docx (Authoritative)
https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/amqp-cbs-v1.0.html
https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/amqp-cbs-v1.0.pdf

Technical Committee:
OASIS Advanced Message Queuing Protocol (AMQP) TC

Chairs:
Rob Godfrey (rgodfrey@redhat.com), Red Hat
Clemens Vasters (clemensv@microsoft.com), Microsoft

Editor:
Clemens Vasters (clemensv@microsoft.com), Microsoft

Related work:
This document is related to:

- *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 0: Overview*. Edited by Robert Godfrey, David Ingham, and Rafael Schloming. 29 October 2012. OASIS Standard. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html.

Abstract:
This specification describes an AMQP authorization mechanism based on claims-based security tokens.

Status:
This document was last revised or approved by the OASIS Advanced Message Queuing Protocol (AMQP) TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=amqp#technical.

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/amqp/.

terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/amqp/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:
When referencing this document, the following citation format should be used:

[AMQP-CBS-v1.0]

AMQP Claims-based Security Version 1.0. Edited by Clemens Vasters. 17 March 2021. OASIS Committee Specification Draft 01. https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/csd01/amqp-cbs-v1.0-csd01.html. Latest stage: https://docs.oasis-open.org/amqp/amqp-cbs/v1.0/amqp-cbs-v1.0.html.

Notices:

Table of Contents

1 Introduction

This specification defines a claims-based security (CBS) extension of AMQP for authorizing interactions with the resources inside an AMQP 1.0 **[AMQP]** container.

The goals for this extension are:

1. To support fine-grained claims-based access control for interactions with the resources inside a container within the scope of an AMQP connection.
2. To work with existing AMQP client libraries without low-level changes.

To satisfy these goals, a layered protocol is defined to place security tokens into a token cache over an AMQP connection. In addition, a new SASL mechanism is introduced for optionally loading this token cache at connection time.

It is assumed that applications will be configured out-of-band with the knowledge as to when claims-based security is to be used and what options are supported, e.g., which security token type is to be used.

1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, "NOT RECOMMENDED", “MAY”, and “OPTIONAL” in this document are to be interpreted as described in BCP14 **[RFC2119] [RFC8174]** when, and only when, they appear in all capitals, as shown here.

CBS node:

The “CBS node” is responsible for accepting tokens to be placed into the token cache. Each container supporting this specification SHOULD provide a CBS node with the address “`$cbs`”. The container MAY override this name with the “cbs-node” connection property.

Claim:

A claim is asserted information about a subject by an issuer. It can be evaluated during an authorization process to determine access rights to protected resources. A claim is represented as a name-value pair.

Resource:

Any feature of an AMQP container that the partner can interact with via AMQP. Most often, tokens will be scoped to the entirety of the container or to individual nodes, but a token can also be scoped to a particular feature. This mechanism would, for instance, enable for authorization tokens to be issued and applied that unlock access to a particular feature for the partner.

Token:

A token contains one or more claims. It may be digitally signed by the issuer so that it can be verified by the receiver. The tokens in this specification are bearer tokens where possession of the token authorizes the bearer to access the resource indicated by the token. Examples of commonly-used formats and encodings for tokens include JSON Web Token (JWT) **[RFC7519]**, Security Assertion Markup Language (SAML) **[SAMLCore],** and Simple Web Token (SWT) **[SWT]**. This specification does not mandate a particular token type other than that it is expressible as a string.

Token Cache:

A connection-scoped token store inside the AMQP container managed through the CBS node.

Token Expiry:

Token expiry is the lifetime (expiration date) for the token after which it must not be accepted. This limits the exposure of the token if it is compromised.

Token Type:

Tokens are assigned a type to allow the receiver to identify the specific claims format and encoding for the token. Types are represented as strings that observe the same namespace convention for types defined in the core AMQP specification. Standard token types have names prefixed with “amqp:”, e.g., the type of a JSON Web Token is represented as “amqp:jwt”. It is recommended that proprietary token types are named using a reverse domain name prefix, e.g., “acme.com:acmetoken”. A registry of commonly defined token types and their meanings is maintained **[AMQPTOKENS]**.

1.2 Normative References

[AMQP]

Godfrey, R., Ingham, D., Schloming, R., “Advanced Message Queuing Protocol (AMQP) Version 1.0”, October 2012. OASIS Standard. https://www.oasis-open.org/standards#amqpv1.0

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997. <http://www.rfc-editor.org/info/rfc2119>

[RFC3339]

Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002. <http://www.rfc-editor.org/info/rfc3339>

[RFC3629]

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003. <http://www.rfc-editor.org/info/rfc3629>

[RFC4301]

Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005. <http://www.rfc-editor.org/info/rfc4301>

[RFC4422]

Melnikov, A., Ed., and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006. <http://www.rfc-editor.org/info/rfc4422>

[RFC5234]

Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008. <http://www.rfc-editor.org/info/rfc5234>

[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>.

[RFC7519]

Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015. <http://www.rfc-editor.org/info/rfc7519>

[RFC8174]

Leiba, B., “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words”, RFC 8174, DOI 10.17487/RFC8174, May 2017. <http://www.rfc-editor.org/info/rfc8174>

[OpenID]

Sakimura N., Bradley, J., Jones, M., de Medeiros, B., and Mortimore, C., OpenID Foundation, "OpenID Connect 1.0", 2014, <https://openid.net/connect/>

[SAMLCore]

S. Cantor et al. “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0”, OASIS SSTC, March 2005. Document ID samlcore-2.0-os. <http://www.oasis-open.org/committees/security/>

[SWT]

Hardt D., Goland Y., “Simple Web Token (SWT)”, November 2009. http://msdn.microsoft.com/en-us/library/windowsazure/hh781551.aspx

1.3 Non-Normative References

[AMQPTOKENS]

AMQP Capabilities Registry: Token Types. http://www.amqp.org/amqp-cbs/1.0/token-types

2 Overview

While some message brokers maintain internal account databases and manage local access control lists for the resources they offer, modern cloud platform systems typically separate identity management and authentication, authorization management, and resource interactions into distinct system services, with obvious benefits:

- User and service identities can be centrally managed and are usable with many different services.
- Authorization permissions or roles can be managed with consistent APIs and user experiences across multiple services, also allowing for standardized roles and permissions that apply uniformly to similar features.
- The burden of processing authentication and authorization requests is offloaded to distinct services, and the result of the authorization action is captured into reusable tokens that can be efficiently evaluated by the service providing the desired resources.

Some systems use the OpenID Connect [OPENID] to interact with authentication services. The resulting identity tokens might then be passed on to an OAuth 2.0 [OAUTH2] authorization service which issues JWT [JWT] access tokens for the desired resource to the client.

In other systems, the identity is established through some out-of-band method and manifests in the client being in possession of a key or key pair, sometimes held inside a signed X.509 [X509] certificate. The key material is then used to sign authorization requests, and the authorization server might return a SAML [SAML] token.

In either model, the resulting authorization token is an opaque character sequence that the client typically does not need to understand. Any metadata that the client needs for follow-up work, such as the instant at which the token expires and must be replaced, is typically provided separately such that the client can understand it without having to parse the token.

In systems like these, the resource service – here an AMQP container – and the authorization service will generally have a private trust relationship that has been established via some out-of-band method, and that relationship typically manifests in the resource service and the authorization service agreeing on a token format and sharing key material that allows the resource service to validate the tokens issued by the authorization service.

When using CBS, an AMQP container maintains a connection-scoped token cache. The CBS mechanism is used to transfer tokens into this cache, either at connection time via SASL or later via a link to the CBS node. Expiring tokens can be replaced via the latter mechanism.

Whenever the connected partner later interacts with resources inside the container, like creating links, the container consults the token cache, locating a token that corresponds to the given resource, and determines whether the token is valid and contains an authorization claim that permits the desired operation.

The CBS mechanism makes no prescriptions for how the tokens relate to resources inside of the AMQP container.

Links that have been established based on the evaluation of a token from the token cache SHOULD be terminated when the token expires and the token cache does not hold a replacement token at the time of expiration. For durable termini, termination of a link due to token expiration has the same effect as the link being closed for any other reason and MAY therefore trigger expiration.

2.1 Interaction Model

CBS composes with the security model defined in **[AMQP]**. An AMQP connection that uses CBS can be established without any authentication context, or the partners can mandate authentication at the transport level and/or use a supported SASL mechanism.

The basic interactions are illustrated in Figure 1:

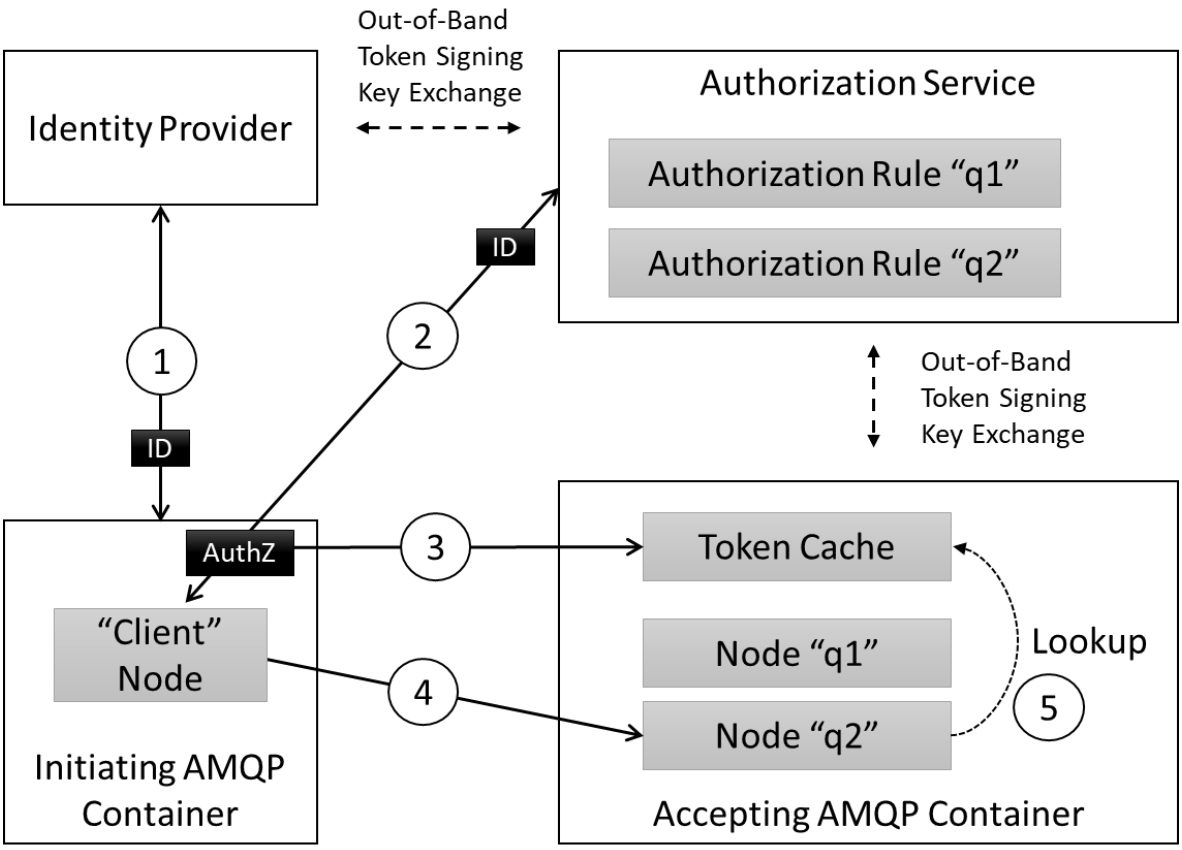

Figure 1: Overview

This illustration is discussing granting access to create a link to a node as an example. A token flow that might grant access to a feature will look similar.

The first two interactions shown in the illustration occur outside of the scope the interactions defined in this specification and merely show how a token might be acquired in a common environment. The token might instead be created through entirely different means.

In step (1), the application that later acts as the connection-initiating AMQP container establishes its identity with an identity server through some authentication mechanism and retrieves an identity token.

In step (2), the application server then takes the identity token to an authorization server, together with an identifier for the resource in the authorization target that it wants to access. The authorization server has an established trust relationship with the identity server such that it is capable of determining whether the token is valid. The authorization server also has a trust relationship with the authorization target – the AMQP container in our case – that will later allow the target to validate its issued tokens. The authorization server is assumed to maintain a rule that allows it to determine whether the given identity has been granted access to the requested resource inside the target and will issue a corresponding access token if that is the case.

Step (1) might be using the OpenID Connect 1.0 [OpenID] authentication protocol. Step (2) might be using the OAuth 2.0 [RFC6749] authorization framework and protocol.

Step (3) is defined in this specification. One or more authorization tokens, each granting some level of access to resources in the AMQP container, are being placed into the container’s token cache. This might occur at connection time via the SASL mechanism defined herein or might occur after the connection is established via a transfer to the CBS node.

Step (4) is a regular interaction between two AMQP containers, whereby the accepting AMQP container demands authorization for particular operations. When the initiating container attempts one of those, it

consults the token cache, looks for a token matching the desired operation and permits it if a matching token is found. This specification does not prescribe how that match is performed.

Because CBS is connection-scoped, interaction with resources secured using CBS must happen over the same connection over which tokens were placed either via SASL or the CBS node. When the connection terminates, the token cache and all associated tokens are dropped.

For reattaching links after a connection has been terminated, the client MUST again provide a set of tokens on the new connection. Those tokens MAY stem from a client-held cache.

The claims-based security mechanism defines an optional, custom SASL mechanism **[RFC4422]**, “AMQPCBS”, that allows seeding the connection with an initial set of tokens. AMQPCBS optimizes the initial handshake, allowing one or more tokens to be set on the CBS node as the connection is created. If the AMQPCBS SASL mechanism is advertised and selected, then subsequent AMQP set-token messages may add to or replace tokens initially seeded using the SASL handshake. Using SASL AMQPCBS also allows for the connection to be protected by tokens rather than requiring some additional connection-level credential verification.

2.2 Client Model

This document does not define either an authentication or authorization protocol nor does it impose any restrictions on protocol choices other than requiring a minimal set of inputs and outputs.

The assumption made for the CBS mechanism is that the client programming model encapsulates the token acquisition with a “token provider” abstraction.

The input to the token provider is

1) an AMQP URL that identifies the container and the resource inside the container for which access is requested
2) a maximum duration for the validity of the acquired token

The output from the token provider is

1) opaque access token string
2) a UTC timestamp indicating the expiration of the token

Since the CBS mechanism allows to replace tokens for links that have already been established, the client SHOULD track the expiration times of tokens it has placed into the token cache and SHOULD acquire a new token before the prior token expires and place the replacement into the cache.

The token provider model as an abstraction allows for client implementations to perform that acquisition silently for as long as the authentication proof or authorization refresh token is valid.

2.3 Scenarios

Link-based and Message-based communication scenarios are basic use cases for claims-based security, but other potential scenarios may exist.

2.3.1 Link-based

In this scenario, a single link is being used to exchange messages with a single endpoint and access to this endpoint is controlled by claims-based security. To be able to exchange messages over a link to that endpoint, an appropriate valid claim is required to be in place.

For example, a message broker hosting a queue with address “q1”, could require a “receive” permission claim for receiving messages and a “send” permission claim for sending messages. In this example, a client would need to set a token containing the appropriate claim for the audience URI identifying “q1” on the CBS node in advance of establishing the link to “q1”. Periodically, before the token expires, the client would need to put a refreshed token on the CBS node for the client and the audience URI identifying “q1” to be able to continue to exchange messages.

The location of the claim(s) and the audience URI or identifier in the token as well as the signature mechanism are application specific.

2.3.2 Message-based

In this scenario, a single link is being used to exchange messages with multiple endpoints. This is referred to as a router scenario as defined in section 2.2 of the AMQP Anonymous Terminus specification **[AMQP-AT]**.

For example, consider a message broker hosting queues with addresses “q1” and “q2” and a routing endpoint with address “relay”. To send messages to queues “q1” and “q2”, a client establishes a link with a target address “relay” and uses the “to” property of messages to specify the desired final address, “q1” or “q2”.

In this example, the broker may require “send” claims for the “relay” as well as for the final destination queues in order to accept a message from the client. Conversely, the broker may secure just the relay or just the final destination queues. It is assumed that the client is aware of what claims are required through some out-of-band configuration.

In this example, if the relay is secured, then the sender would need to set a token containing the appropriate claim along with the relay’s address covered by the audience URI on the CBS node in advance of establishing the link. Periodically, before the token expires, the client would need to send a refreshed token to be able to continue to exchange messages with the relay. In addition, the sender would need to set appropriate tokens with the audience URI covering each target endpoint referenced in the “to” addresses of messages sent via the relay in advance of sending a message.

3 Managing the Token Cache

Tokens are communicated between AMQP partners by transferring well-defined AMQP messages to the CBS node which manages the token cache on a per-connection basis. Tokens can only be set; they cannot be changed or deleted. If there is a need to delete tokens, the sending party can drop the connection, which instantly clears the associated cache.

3.1 Connection Capability

On connection establishment, a partner MUST indicate whether it supports claims-based security through the exchange of connection capabilities (see Section 2.7.1 [**[AMQP]**]).

Capability Name	Definition
AMQP_CBS_V1_0	If present in the `offered-capabilities` field of the `open` frame, the sender of the `open` supports the use of claims-based security by its receiver. If present in the `desired-capabilities` field of the `open` frame, the sender of the `open` MUST use claims-based security if the receiver of the `open` supports this capability.

The container MAY tell the partner the address of the CBS node with a connection property:

Connection property name	Type	Definition
cbs-node	address	Address of the CBS node. If omitted or null-valued, the CBS node address defaults to $cbs.

The container offering the AMQP_CBS_V1_0 capability either MUST provide a CBS node address in the connection property or it MUST use the reserved name $cbs for the CBS node.

3.2 Establishing a Link

The link for communicating tokens to the CBS node is established with the CBS node address as target.

An implementation MAY make access to the CBS node address conditional on a lower-level access control mechanism, for instance it may require having established an authorized SASL authentication context.

The lower-level authorization context MAY be established with SASL ANONYMOUS, granting anyone permission to set tokens on the CBS node with the intent of establishing CBS-scoped authorization contexts.

If allowing anonymous access, an implementation SHOULD constrain the time during which the connection and the link may exist without any valid token havening been set.

Messages MAY be routed to the CBS node address via the Anonymous Terminus, under the condition that the routing link is established in conformance with the rules laid out in this section.

In the `attach` frame for the *sender* `role` from the Client:

- the `snd-settle-mode` field SHOULD be set to *unsettled*
- the `rcv-settle-mode` field MUST be set to *first*
- the `outcomes` field of the `source` field MUST contain *amqp:accepted:list* and *amqp:rejected:list* which are the only outcomes supported by the CBS Node

In the `attach` frame for the *receiver* `role` from the CBS Node:

- the `rcv-settle-mode` field MUST be set to *first*
- the `durable` field of the `target` field MUST be set to *none*. (The CBS Node does not support link resumption)

3.3 Setting a Token

A token is cached on the CBS node by transferring a “set-token” message.

The assumption made here is that the token cache understands the token format and can therefore take all relevant information about what audience the token applies to and when the token expires from the token itself, and therefore also whether a given token is replacing an existing one.

3.3.1 set-token Message

The *subject* property of the set-token message MUST be set to the “set-token” string.

In the application-properties section, the message SHOULD carry a property named “token-type”, with a string value that indicates the type of the token carried inside the payload. The token type is typically known to the client based on the authorization service it interacts with. The value MAY be omitted, but the token cache MAY reject tokens it cannot understand without this hint.

The body of the message MUST contain the token as an AMQP Value, and will typically be a string.

3.3.2 Indication of Settlement

If the request is successful, the CBS Node MUST respond to the sender with a disposition outcome of `accepted`.

If the request is unsuccessful due to a processing error, the CBS node MUST respond to the sender with a disposition outcome of `rejected`. Further information MAY be provided in the `error` field in the `rejected` outcome.

For error conditions related to the content of the request, e.g., unsupported token type, malformed request etc., an application-specific description MAY be provided in the `error` field, with consideration for general best practice for security-related protocols, meaning a potential attacker ought not to be able to learn information helping to improve their attack.

4 TLS and SASL Integration

The claims-based security mechanism composes with the TLS and SASL security foundation described in **[AMQP]**.

Security tokens used with this mechanism provide access control at a more detailed level than the coarse connection-scoped level that original AMQP security model provides.

Because interception of a valid token could allow an attacker to gain access to the node, this mechanism MUST be used with TLS as defined in **[AMQP]** or the communication path MUST otherwise be protected through lower-level mechanisms such as IPSec **[RFC4301]**.

4.1 Integration with common SASL mechanisms

The claims-based security mechanism can be combined with SASL mechanisms depending on the protection needs.

The partner playing the role of the SASL client and the partner playing the role of the SASL server MUST correspond to the TCP client and server respectively.

4.1.1 SASL ANONYMOUS

The partner acting as the SASL server MAY announce the SASL mechanism *ANONYMOUS* in the `sasl-mechanisms` frame body, allowing the partner acting as the SASL client to establish an anonymous initial connection, a session, and a link with the CBS node.

Allowing *ANONYMOUS* carries the risk of allowing unauthenticated clients to open and maintain (potentially very many) connections with the server, leading to significant resource consumption, which is a potential denial-of-service threat vector.

It is RECOMMENDED that the server only allows anonymous connections to be maintained for the duration required to perform an initial successful CBS `set-token` operation with a verified valid token.

4.1.2 SASL EXTERNAL

The partner acting as the SASL server MAY announce the SASL mechanism *EXTERNAL* in the `sasl-mechanisms` frame body, if the underlying transport session from the partner acting as the SASL client has been established using some form of client authentication, such as TLS with X509 client certificates, TLS with pre-shared symmetric key, or raw-public-key credentials, or IPSec with equivalent credentials.

In this case, the transport session authentication provides protection for the initial connection, session, and link to the CBS node, while the claims-based security mechanism specifically protects access to the Nodes managed by the CBS node.

4.1.3 SASL PLAIN and Others

The partner acting as the SASL server MAY announce the SASL mechanism *PLAIN* or any other SASL mechanism in the `sasl-mechanisms` frame body that is suitable to establish an authenticated context between the partners.

The authentication context provides protection for the initial connection, session, and link to the CBS node, while the claims-based security mechanism specifically protects access to the Nodes managed by the CBS node.

4.2 SASL AMQPCBS Mechanism

The partner acting as the SASL server MAY announce the SASL mechanism *AMQPCBS* in the `sasl-mechanisms` frame body. This mechanism integrates AMQP CBS capabilities into the SASL authentication exchange.

4.2.1 SASL and MIN_MAX_FRAME_SIZE

As defined in Section 5.31 of **[AMQP]**:

> *The maximum size of a SASL frame is defined by MIN-MAX-FRAME-SIZE. There is no mechanism within the SASL negotiation to negotiate a different size.*

Due to the requirement to transfer potentially large sets of tokens during the SASL exchange, implementations of the AMQPCBS SASL mechanism MUST support a maximum SASL frame size of 8192.

4.2.2 SASL Init

After selecting the AMQPCBS SASL mechanism, the partner acting as the SASL client MUST send a SASL Init with *AMQPCBS* selected in the `mechanism` field and response data returned in the `initial-response` field of the `sasl-init` frame body. The response data is a list of tokens, equivalent to an ordered sequence of `put-token` messages as described in **Setting a Token**. At least one token MUST be set.

The formal grammar for the `response` field using ABNF **[RFC5234]** follows:

```
NUL                 = %x00
NON-NULL-UTF8       = *(%x01-7F / UTF8-2 / UTF8-3 / UTF8-4)

TOKEN-TYPE          = NON-NULL-UTF8 NUL
TOKEN-VALUE         = NON-NULL-UTF8 NUL

TOKEN               = TOKEN-TYPE TOKEN-VALUE

RESPONSE            = 1*TOKEN NUL NUL
```

The response data is an ordered list of tokens. Each token in the list is composed of a token type and a token value. All elements are encoded as UTF-8 strings **[RFC3629]** followed by a NUL (%x00) character. There is no special row separator.

If the response data contains the complete list of tokens, then the last token is followed by two NUL characters. The partner acting as the SASL server MUST then send a SASL Outcome.

If the list of tokens exceeds the MIN-MAX-FRAME-SIZE, additional SASL Challenge-Response pairs MUST be exchanged until all tokens have been received or an error occurs. Then the partner acting as the SASL server MUST send a SASL Outcome.

The primary scenario for the AMQPCBS SASL mechanism is seeding a token cache. The tokens MAY be validated as they are set, and if validation is performed and it fails, the SASL Outcome MUST indicate that failure.

4.2.3 SASL Challenge

In response to receiving a partial list of tokens in the SASL Init or SASL Response, the partner acting as the SASL server MUST send a SASL Challenge with no challenge security data in the `challenge` field of the `sasl-challenge` frame body to continue the exchange. Otherwise, it MUST send a SASL Outcome indicating that the exchange was unsuccessful as described in **SASL Outcome.**

4.2.4 SASL Response

After receiving the SASL Challenge, the partner acting as the SASL client MUST send a SASL Response. The response data in the `response` field of the `sasl-response` frame body contains a list of tokens, equivalent to an ordered sequence of `put-token` messages as described in **Setting a Token**.

The formal grammar and the response data for the `response` field is the same as described in **SASL Init**.

If the response data contains the remaining tokens, then the last token is followed by two NUL characters. The partner acting as the SASL server MUST then send a SASL Outcome; otherwise, additional SASL Challenge-Response pairs MUST be exchanged until all tokens have been received or an error occurs. Then the partner acting as the SASL server MUST send a SASL Outcome.

4.2.5 SASL Outcome

When the SASL exchange is complete or an error occurs, the partner acting as the SASL server MUST send a SASL Outcome, indicating the outcome in the `code` field in the `sasl-outcome` frame body. If the exchange is successful, the code field MUST be set to 0; otherwise, the code field MUST be set to one of the failure codes defined by the `sasl-code` type. No data is returned in the `additional-data` field in the `sasl-outcome` frame body.

5 Conformance

When considering this specification, we can consider two distinct roles an AMQP container may play:

Firstly, that of an Initiating Container– a container which wants to initiate and perform operations on resources requiring authorization; Secondly an Accepting Container – a container which offers such resources.

An Initiating Container is conformant with this specification if:

1. Upon selecting the AMQPCBS SASL method, the SASL handshake is performed as per section 4.2
2. All CBS interactions defined in sections 3.2 and 3.3 are performed as specified.
3. Upon being offered the ”cbs-node” connection property, all interactions are performed via the indicated address instead of the “$cbs” default address.

An Accepting Container is conformant with this specification if:

1. Upon offering the AMQPCBS SASL method, the SASL handshake is performed as per section 4.2
2. Upon offering the connection capability defined in section 3.1, the container accepts set-token messages as defined in section 3.3 on the default or connection-property indicated node address.

Because the CBS mechanism makes no prescriptions for how the tokens relate to resources inside of the AMQP container, there are no conformance rules defined related to how authorization is performed using the token cache.

6 Security Considerations

The CBS mechanism is a security mechanism that introduces two key alternatives to other AMQP authentication and authorization methods that are worth highlighting:

1. The authorization scope moves from the connection level to the resource level. While this specification does not prescribe a relationship between tokens and resources, resource access MUST be guarded by consulting the token cache at the resource level. It is no longer guarded by an access check at the connection level.
2. If the AMQPCBS SASL mechanism is used, no other SASL mechanisms can be combined with it, therefore all implementations that rely on establishing a security context using other mechanisms must be able to deal with an anonymous context at the connection level or the connection itself must be authorized by a token transferred using the AMQPCBS SASL handshake.

The CBS mechanism can be implemented by first establishing an anonymous context for the connection using the ANONYMOUS SASL mechanism and for the initiator to subsequently transfer tokens into the cache. In this case, it is RECOMMENDED for the anonymous initiator to only be allowed to communicate with the CBS node and for the time window from establishing the connection to transferring such tokens and to create the initial non-CBS link to be short.

Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Alan Conway, Red Hat
Keith Wall, Red Hat
Robbie Gemmell, Red Hat
Justin Ross, Red Hat
Ted Ross, Red Hat
Oleksandr Rudyy, JP Morgan
Xin Chen, Microsoft
Clemens Vasters, Microsoft

Appendix B. Revision History

Revision	Date	Editor	Changes Made
WD03	March 31 2017	Clemens Vasters	Added TLS and SASL Integration Added AMQPCBS SASL Mechanism
WD03	March 31 2017	Brian Raymor	Updated Normative References Moved Concepts to Terminology section Rewrote CBS interactions to use Disposition Added Connection Capability for CBS Drafted ABNF for SASL Mechanism Updated AMQPCBS SASL Mechanism to support multiple challenge-response exchanges
WD04	July 27 2017	Brian Raymor	**AMQP-100** Increasing MIN-MAX-FRAME-SIZE for SASL AMQPCBS **AMQP-101** Added amqp:sasl as standard token type **AMQP-102** Detailed descriptions for error conditions related to content **AMQP-103** Removed Type field from delete-token **AMQP-104** SASL Outcome: differentiating application-data based on code **AMQP-105** AMQPCBS: Indicating that multiple challenge-responses are required to transmit token set **AMQP-107** Clarifying definition for Token Name (audience)? **AMQP-115** "Type" should be "type" in put-token and delete-token application-properties **AMQP-116** Clarify: putting the same token multiple times **AMQP-119** Deleting tokens **AMQP-120** Make the address "$cbs" more explicit **AMQP-122** Are there any restrictions on token names Updated Terminology section to include the RFC8174 BCP update
WD05-WD08	May 23 2019	Clemens Vasters	TBDUpdates based on AMQP Face-to-Face Meeting in Berlin
WD10	June 25 2020	Clemens Vasters	External reference cleanup
WD11	October 2, 2020	Clemens Vasters	Clarifying edits

Appendix C. Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website: [https://www.oasis-open.org/policies-guidelines/ipr].

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OASIS AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THIS DOCUMENT OR ANY PART THEREOF.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specifications, OASIS Standards, or Approved Errata).

[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this deliverable.]

[OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website, but disclaims any obligation to do so.]

[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this OASIS Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]

The name "OASIS" is a trademark of OASIS, the owner and developer of this document, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, documents, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.