ONEM2M TECHNICAL REPORT

Document Number	oneM2M-TR-0019-V-2.0.0
Document Name:	Dynamic Authorization for IoT
Date:	2016-December-07
Abstract:	This technical report analyzing the requirements for, and elaborating, dynamic authorization solutions suitable for open IoT scenarios

Contents

1 Scope

Development of the Internet of Things implies convenient ways to enable interactions between M2M Applications depending on different actors, and possibly affiliated to different M2M Service Providers. In the context of an M2M Service Platform, this relies on a practical means for a Resource consuming application to obtain possibly temporary and restricted access to a Resource exposed by a Resource producing application. The Access Control Policies mechanism of oneM2M TS-0003 [i.3] was suitable under the assumption of centralized (client-server style) M2M deployments where interactions between Resource producers and Resource consumers are mainly predictable at the time of resource creation and restricted within a known (reduced) set of interacting entities that is not constantly evolving. This assumption is no longer valid in IoT scenarios where many-to-many application level interactions between multitudes of devices would result in exponential explosion in the number of Access Control Policies to establish and manage, while their unpredictable and fast evolving nature would create a bottleneck where the ACPs are maintained and evaluated.

This work item will therefore investigate the suitability of alternative authorization schemes. This can include e.g. token-based models, where the grant of authorization is directly embodied by a virtual ticket delivered to the requester that carries a scope of authorization as well as validation means.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] oneM2M Drafting Rules.

NOTE: Available at http://www.onem2m.org/images/files/oneM2M-Drafting-Rules.pdf.

[i.2] oneM2M TS-0001: "Functional Architecture".

[i.3] oneM2M TS-0003: "Security Solutions".

[i.4] oneM2M TS-0011: "Common Terminology".

[i.5] oneM2M TR-0016: "Study of Authorization Architecture for Supporting Heterogeneous Access Control Policies".

[i.6] IETF RFC 6749 (2012): "The OAuth 2.0 Authorization Framework".

[i.7] IETF draft-hardjono-oauth-umacore-13: "User Managed Access Profile of OAuth 2.0".

NOTE: Available at http://tools.ietf.org/html/draft-hardjono-oauth-umacore-13.

[i.8] OpenId foundation. OpenID Connect Dynamic Client Registration 1.0. [Online].

NOTE: Available at http://openid.net/specs/openid-connect-registration-1_0.html.

[i.9] oneM2M TS-0004 "Service Layer Core Protocol Specification".

[i.10] IETF RFC 7519 (2015): "JSON Web Token (JWT)".

[i.11] IANA JSON Web Token (JWT) registry.

NOTE: Available at http://www.iana.org/assignments/jwt/jwt.xhtml.

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in oneM2M TS-0011 [i.4], oneM2M TS-0003 [i.3] and the following apply:

access token [i.6]: dynamically issued credential which can be used to access resources

NOTE: An access token represents an authorization issued to the Originator.

access token issuing privileges: those privileges that the Token Authority is allowed to delegate to other entities by issuing Access Tokens

authorization grant [i.6]: dynamically issued credential representing the Grant Approver's authorization to access resources

NOTE: An authorization grant can either be used directly access token, or used by the Originator to obtain an access token.

authorization server: entity with permission to access tokens to the originator upon presentation of an authorization grant. Performs a functional role similar to an OAuth 2.0 authorization server [i.6]

dynamic authorization: enables a service provider or resource owner with the ability to provide access privileges, based on one or more authorization checks to an entity such that it is able to perform operations on resource(s) for a finite duration, when a prior relationship or static access control policy does not exist for the operation requested by that entity

(oneM2M) dynamic authorization architecture: framework providing dynamic authorization for obtaining access to resources

grant approver: entity with permission to grant access to a protected resource

NOTE: Provides part of the functionality of an OAuth 2.0 resource owner [i.6].

grant issuer: entity recognized (by the Host CSE and/or an authorization server) as having permission to issue authorization grants

NOTE: Provides part of the functionality of an OAuth 2.0 resource owner [i.6].

3.2 Symbols

For the purposes of the present document, the symbols given in oneM2M TS-0011 [i.4] and oneM2M TS-0003 [i.3] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in oneM2M TS-0011 [i.4], oneM2M TS-0003 [i.3] and the following apply:

AS	Authorization Server
DAA	(oneM2M) Dynamic Authorization Architecture
Daga	DAA authorization grant approval reference point
Dagd	DAA authorizationg grant data reference point
Dagi	DAA authorization grant issuance reference point
Datd	DAA access token data reference point
Dati	DAA access token issuance reference point
Datu	DAA access token usage reference point
OAuth	Web Authorization Protocol

NOTE: Denotes an IETF Working Group and specifications produced by that working group.

RO	Resource Owner
RPT	Requesting Party Token
RS	Resource Server
SLDA	Service Layer Dynamic Authorization
SLDA-PA	Service Layer Dynamic Authorization Policy Administration
SLDA-PD	Service Layer Dynamic Authorization Policy Determination
SLDA-PE	Service Layer Dynamic Authorization Policy Enforcement
SLDA-PI	Service Layer Dynamic Authorization Policy Information
UMA	User Managed Access

4 Conventions

The key words "Shall", "Shall not", "May", "Need not", "Should", "Should not" in the present document are to be interpreted as described in the oneM2M Drafting Rules [i.1].

5 Use Cases

5.1 Use Case for Access Token in Dynamic Authorization

5.1.1 Description

In a oneM2M System the Platform, Gateways and Devices interact with each other in the way of many-to-many, furthermore many of these relations can be dynamic and temporary. It is desirable that the access control mechanism used by the M2M System is flexible and efficient.

The use case in the following clauses shows the use of dynamically issued access tokens that carry authorization information to simplify the access control management. In this use case the mobile phone of a tourist is issued with access tokens in which there are access privileges to the resources of M2M Gateways/Devices. The tourist then can use these access tokens to access various M2M Gateways or Devices in the hotel or attractions, such as accessing his/her room, operating electric devices in the room, accessing the VIP area, using the facilities in the fitness center, renting a bicycle, or using some tourist facilities in the attractions. If using the traditional way, i.e. the privileges are placed in the local stored access control policies, then all the access control policies in the gateways or devices related to this tourist are expected to be updated. As there are so many tourists coming and going, the workload of modifying and uploading access control policies will be very large, if not impossible.

5.1.2 Actors

The entities involved in this use case are shown in figure 5.1.2-1 and described as follows:

M2M Platform: It represents an infrastructure entity that interacts with M2M Gateways/Devices and M2M Application to provide common functionalities. In this use case the M2M Platform provides communication path between two M2M Gateways/Devices that do not have registration relation.

Token Issuer: It represents an M2M Gateway that is responsible for issuing access tokens to other M2M Gateways/Devices.

Mobile Phone: It represents an M2M Device held by a tourist. Access tokens are issued to the mobile phone, and the tourist uses the mobile phone to interact with other M2M Gateways/Devices.

M2M Gateway: It is responsible for caching and forwarding the messages exchanged between the M2M Platform and target M2M Devices. It also acts as a control center that stores various information related to the M2M Devices that have registered to it.

M2M Device: It represents various M2M devices such as door lock of door, fitness equipment, bicycle lock, smart TV, facilities for tourists at attractions and so on.

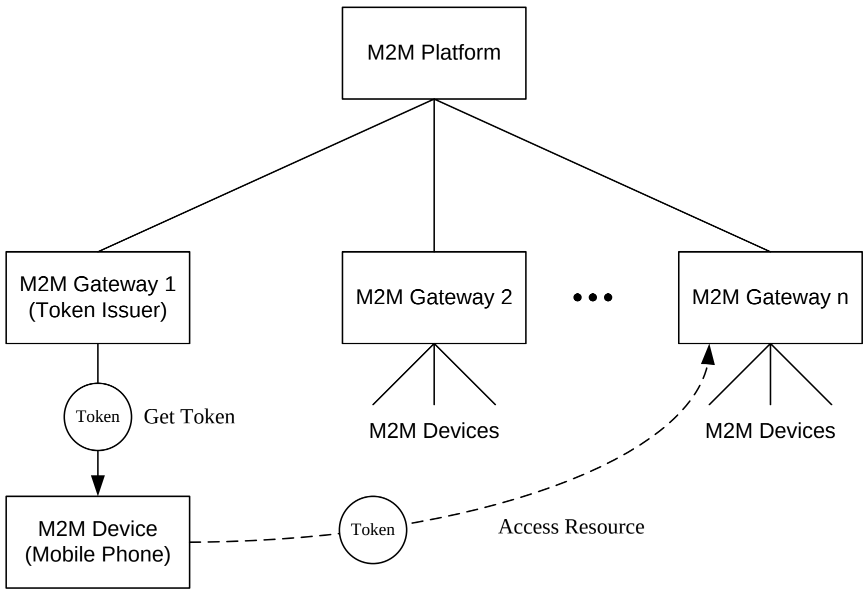


Figure 5.1.2-1: Using token to carry authorization information

5.1.3 Pre-conditions

The issuer (M2M Gateway 1) of access tokens is provisioned with security credentials that are used to provide integrity and confidentiality protection for access tokens.

Other M2M Gateways are provisioned with security credentials that are used for verifying received access tokens.

5.1.4 Normal Flow

The procedure of issuing and using access tokens is:

1) A tourist registers his/her mobile phone as an M2M Device with the M2M Gateway 1 at the hotel reception. The M2M Gateway 1 also plays the role of token issuer.

2) The token issuer issues one or multiple access tokens to the mobile phone according to the services booked by the tourist. Inside the tokens there is some authorization information describing what privileges are assigned to the mobile phone (M2M Device).

NOTE: The tourist can then use these access tokens to access various M2M Gateways or Devices in the hotel through his/her mobile phone, such as accessing his/her room, operating electric devices in the room, accessing the VIP area, using the facilities in fitness center, renting a bicycle, or using some tourist facilities in the attractions.

3) When the tourist arrives at some sites such as room door, the mobile phone detects an M2M Gateway or an M2M Device with which it can make contact. The mobile phone then establishes security association with the detected device via the M2M Gateway 1 and M2M Platform. After that a resource access request with appropriate access tokens are sent to the M2M Gateway/Device.

4) The M2M Gateway/Device verifies the access tokens sent by the mobile phone, extracts the privileges from the token, and evaluates the access request with the privileges. If it is permitted, the M2M Gateway/Device executes the requested operation on the target resource.

5.1.5 Potential requirements

1) The authorized M2M entity shall be able to issue access tokens that describe what privileges are assigned to the holder of the access tokens used to access the resources in other M2M entities.

2) The M2M System shall provide authenticity, integrity and confidentiality protection for access tokens.

3) The M2M System shall be able to revoke the access tokens that are not yet expired.

5.2 Use Case for Role Token in Dynamic Authorization

5.2.1 Description

According to the description in oneM2M TS-0001[i.2], in the oneM2M System the M2M Service Subscription defines the technical part of the contract between an M2M Subscriber and an M2M Service Provider. An M2M Service Subscription establishes a link between one or more AEs; one or more M2M Nodes, one or more M2M Services. In each M2M Service, one or multiple M2M Service role(s) are defined by the M2M Service Provider. An M2M Service role is mapped to a created permission pertaining to resource types which are associated with M2M Service. The M2M Subscriber subscribes to one or multiple Service role(s) within the M2M Services.

In Role Based Access Control (RBAC) roles are assigned to users, and privileges are assigned to roles, users obtain privileges through their assigned roles. There are two ways to implement the role-user assignments, one is both role-user assignments and privilege-role assignments are described in the RBAC policies; another is only privilege-role assignments are described in the RBAC policies, the user-role binding is achieved in the time of access control. One way or another depends on the specific application scenario. If the role-user assignment is stable, user-role assignments can be described in RBAC policies, otherwise another way can be considered.

In the oneM2M application environment the relations between AEs, CSEs, M2M Nodes and M2M Services can dynamically change. This dynamical changing can bring some access control issues. For example, in the use case shown in the Figure 5.2.1-1, the house owners can subscribe to various M2M Services and can also change their service subscriptions from time to time. When the house owners change their Service Subscriptions, it can result in a large number of access control policy revision if the user-role assignments is implemented in RBAC policies. Also note that in many IoT scenarios the definition of static Roles is not as straightforward as in corporate use cases where this concept was most successful. So in the oneM2M System security mechanisms or approaches are considered to address such issues.

The ABAC model, which considers contextual properties in addition to Originators and target resources, enables some improvements to manage the increasing complexity. However several IoT use cases rely on dynamicity in the assignment of Roles and the management of Access Control Policies would remain a burden in dynamic scenarios.

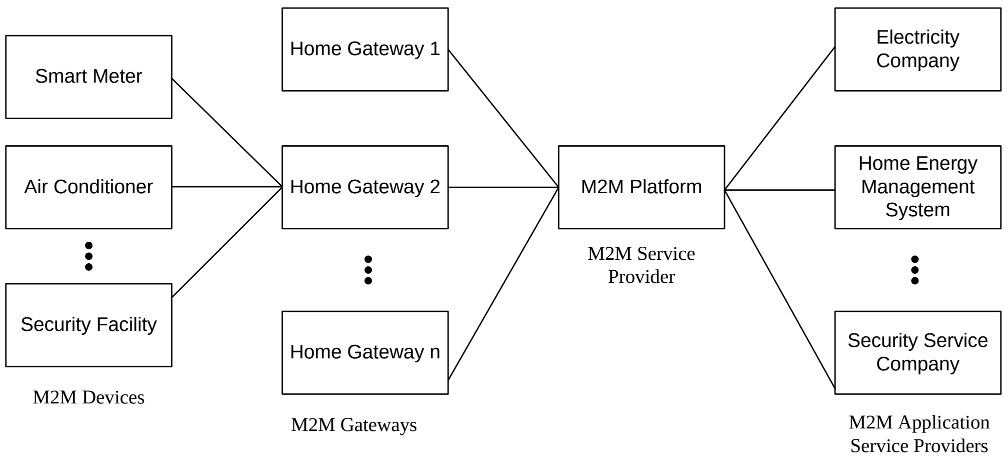


Figure 5.2.1-1: Home Facility Management System High Level Illustration

The use case in the following clauses shows using dynamically issued security tokens to carry Service Roles in order to avoid modifying RBAC policies at the CSEs in the Field Domain.

5.2.2 Actors

The entities involved in this use case are shown in the Figure 5.2.2-1 and described as follows:

M2M Platform: It represents an infrastructure entity that interacts with M2M Gateways/Devices and M2M Application to provide common functionalities for the M2M Services such as collecting the status and configuration information of home devices and controlling them via the home gateway.

M2M Application: It provides some M2M Services for the users through the M2M Service Platform.

EXAMPLE: A Home Energy Management System (HEMS) provides management services for home electronic equipment to minimize energy consumption, and a Security Service Company provides home security related services such as room/house monitoring, fire monitoring and intrusion monitoring.

M2M Gateway: It is responsible for caching and forwarding the messages exchanged between the M2M Platform and target M2M Device. It also acts as a house control center that stores various information pertaining to M2M Devices in the house, such as smart meter data that is waiting for upload, air conditioner control policy, security monitoring records, and so on.

M2M Devices: They represent various M2M devices such as smart meter, air conditioners, fire alarms, room/house monitors and so on.

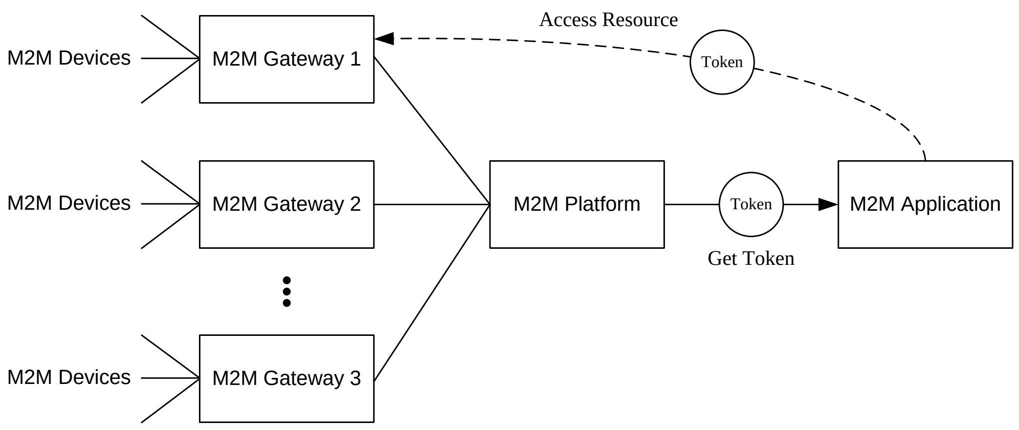


Figure 5.2.2-1: Using token to carry Service Roles in RBAC

5.2.3 Pre-conditions

The M2M Platform is provisioned with security credentials that are used to provide integrity and confidentiality protection for access tokens.

M2M Gateways are provisioned with security credentials that are used to verify received access tokens.

5.2.4 Normal Flow

The procedure of issuing and using access tokens is:

1) A M2M Application that has registered with the M2M Platform sends a Service Role Token Request to the M2M Platform.

2) The M2M Platform first authenticates the M2M Application, and then checks the Service Role Token issuing policies. If the issuance is permitted, the M2M Platform issues a Service Role Token to the M2M Application. This token is protected using the provisioned security credentials.

3) The M2M Platform sends the generated Service Role Token to the M2M Application, or the Service Role Token is retrieved by the M2M Application later.

4) The M2M Application sends a resource access request to the M2M Gateway. In the request the Service Role Token is included.

5) The M2M Gateway authenticates the M2M Application, verifies the Service Role Token using the provisioned security credentials, extracts the Service Roles from the Service Role Token, and evaluates resource access request with applicable RBAC policies. If this access is permitted, the M2M Gateway executes the requested operation on the target resource.

5.2.5 Potential requirements

1) The oneM2M System shall support Role Based Access Control.

2) The M2M System shall be able to issue access tokens for carrying role information in order to facilitate Role Based Access Control.

3) The M2M System shall provide authenticity, integrity and confidentiality protection for access tokens.

4) The M2M System shall be able to revoke the access tokens that are not yet expired.

5.3 Use case of Dynamic Authorization Policy Provisioning

5.3.1 Description

In a oneM2M System the Platform, Gateways and Devices interact with each other in the way of many-to-many, furthermore many of these relations can be dynamic and temporary. The access control mechanism used by the M2M System are expected to be flexible and efficient and authorization policies that support such dynamic authorization are expected to be tailored accordingly and provisioned to the various authorization components.

In this use case, provisioning of dynamic authorization policies to the various dynamic authorization components is described. An administrator that is authorized uses the Dynamic Authorization Policy Retrieval Point (DA-PRP) to define, create and modify appropriate dynamic authorization policies associated with each of the dynamic authorization components. The DA-PRP then provisions the appropriate policies to the DA-PDP, DA-PEP and information to the DA-PIP on a regular basis.

The present use case assumes that the PRP would be provided by the M2M Platform while the PDP, PEP and PIP would be co-located in the M2M Gateway. Though this can make sense in some use cases, such choice would consider the involved complexity to manage security policies. In the many IoT scenarios that involve multiple resource servers of which M2M Gateways are just a particular instance, the management of the authorization policies in each of those distributed entities is likely to become very complex, especially when multiple stakeholders are involved. In such use cases, centralizing the whole policy management in the M2M Platform to implement only the PEP in the resource servers (which can be constrained devices) would be more judicious.

5.3.2 Actors

The entities involved in the use case are shown in Figure 5.3.2-1 and described as follows:

M2M Platform: It represents an infrastructure entity that interacts with M2M Gateways /Devices and the M2M Application to provide common functionalities. In this use case the M2M Platform provides the ability to provision dynamic authorization policies by implementing a Dynamic Authorization - Policy Retrieval Point (DA-PRP).

M2M Gateway: It represents a gateway that is responsible for hosting a resource. In addition, the M2M Gateway is responsible for implementing the Dynamic Authorization - Policy Enforcement (DA-PEP), Dynamic Authorization - Policy Decision Point (DA-PDP) and Dynamic Authorization - Policy Information Point (DA-PIP) functions.

M2M Device: It represents a sensor application or a sensor device that is responsible for measuring sensor data and hosting a resource on an M2M Gateway.

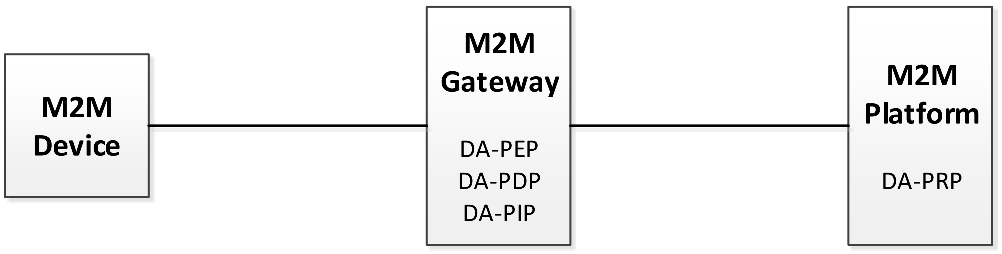


Figure 5.3.2-1: Entities involved in provisioning of Dynamic Authorization policies

5.3.3 Pre-Conditions

The M2M Platform that implements the DA-PRP, has access to a policy database that is configured with dynamic authorization rules.

It is assumed that the communications between the M2M Platform and the M2M Gateway occurs over a secure communications channel.

It is assumed that the communications between the M2M Device and the M2M Gateway occurs over a secure communications channel.

5.3.4 Normal Flow

Procedure for provisioning of dynamic authorization policies:

1) An administrator that has authorized access to the M2M Platform configures appropriate policies that are to be provisioned using a DA-PRP.

2) The DA-PRP at the platform, provisions appropriate decision policies to the DA-PDP, enforcement policies to the DA-PEP and information to the DA-PIP. The provisioning process is performed periodically.

5.3.5 Potential Requirements

The M2M system shall support provisioning of dynamic authorization policies.

5.4 Use case of Dynamic Authorization

5.4.1 Description

In a oneM2M System the Platform, Gateways and Devices interact with each other in the way of many-to-many, furthermore many of these relations can be dynamic and temporary. It is desirable that the access control mechanism used by the M2M System is flexible and efficient.

In this use case an M2M Device would like to perform CRUD operations on resource(s) hosted on an M2M Gateway. The M2M device can be registered with the M2M Gateway or can even be completely unknown to it. Even if the M2M Device is registered to the Gateway, it is deemed that the Device is restricted from being able to perform one or more of the CRUD operations on the resource based on a traditional static access control policy. Dynamic authorization enables a previously restricted M2M Device to be able to perform newer operations on resource(s) hosted at the M2M Gateway. Dynamic authorization checks can be dictated based on dynamic authorization policies which can dictate the types of checks (e.g. higher-level of authentication checks, payment based authorization, platform validation checks etc.) that are performed. The authorization provided can be for a finite period of time and can be added to the static access control policy.

The present use case assumes that the PRP would be provided by the M2M Platform while the PDP, PEP and PIP would be co-located in the M2M Gateway. Though this can make sense in some use cases, such choice would consider the involved complexity to manage security policies. In the many IoT scenarios that involve multiple resource servers of which M2M Gateways are just a particular instance, the management of the authorization policies in each of those distributed entities is likely to become very complex, especially when multiple stakeholders art involved. In such use cases, centralizing the whole policy management in the M2M Platform to implement only the PEP in the resource servers (which can be constrained devices) can be more judicious.

5.4.2 Actors

The entities involved in the use case are shown in Figure 5.4.2-1 and described as follows:

M2M Gateway: It represents a gateway that is responsible for hosting a resource. In addition, the M2M Gateway is responsible for implementing the Dynamic Authorization - Policy Administration Point (DA-PRP), Dynamic Authorization Policy Enforcement (DA-PEP), Dynamic Authorization Policy Decision (DA-PDP) functions and storing Policy Information Point (DA-PIP).

M2M Device: It represents a sensor application or a sensor device that is responsible for measuring sensor data and hosting a resource on an M2M Gateway.

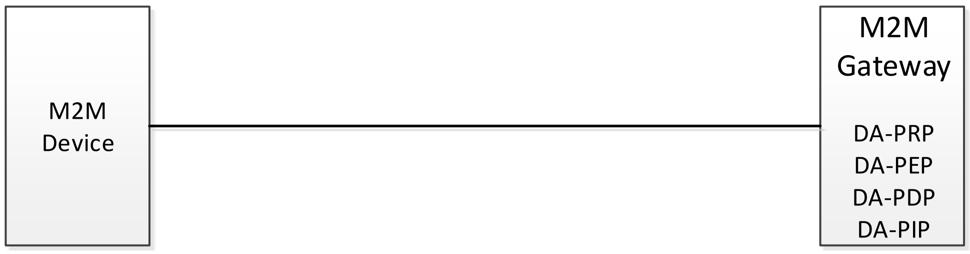


Figure 5.4.2-1: Entities involved in Dynamic Authorization

5.4.3 Pre-Conditions

The M2M Gateway is provisioned with appropriate dynamic authorization policies based on which, it decides on the dynamic authorization checks that will be performed.

It is assumed that the communications between the M2M Device and the M2M Gateway occurs over a secure communications channel.

5.4.4 Normal Flow

Procedure for dynamic authorization:

1) An M2M Device requests to perform a CRUD operation on a resource hosted on the M2M Gateway.

2) The M2M Gateway determines that the M2M Device does not have the appropriate authorization based on a check with the static ACPs.

3) The DA-PDP at the M2M Gateway then consults its dynamic authorization polices in order to determine if the M2M Device can be provided with authorization. If dynamic authorization can be carried out, then the M2M Gateway determines the types of authorization checks that will be performed with the M2M Device.

4) The M2M Gateway in conjunction with the M2M Device performs one or more dynamic authorization checks (e.g. multi-factor authentication checks, platform validation checks, payment-based authorization, subscription, etc.)

5) If the authorization checks have been successfully completed then the DA-PDP updates the relevant entries within the DA-PEP so that the M2M Device is able to perform one or more CRUD operations on a resource for a finite amount of time. Additionally, in certain cases, the DA-PDP can update the ACP to include the M2M Device and the resource/operations that it is allowed to perform based on the dynamic authorization results.

6) The M2M Gateway provides a response to the M2M device indicating the authorization that was provided to it.

5.4.5 Potential Requirements

1) The M2M system shall support that all the dynamic authorization functions reside at the same M2M Gateway.

2) The M2M system shall support various authorization checks (e.g. multi-factor authentication, platform validation, payment authorization etc.) in order to enable dynamic authorization.

3) The M2M system shall enable an authorized entity (e.g. DA-PDP) to update the ACP based on results of the dynamic authorization checks.

5.5 Use case of Dynamic Authorization

5.5.1 Description

In a oneM2M System the Platform, Gateways and Devices with interact each other in the way of many-to-many, furthermore many of these relations can be dynamic and temporary. The M2M System relies on the access control mechanisms being flexible and efficient.

In this use case an M2M Device would like to perform CRUD operations on resource(s) hosted on an M2M Gateway. The M2M device can be registered with the M2M Gateway or can even be completely unknown to it. Even if the M2M Device is registered to the Gateway, it is deemed that the Device is restricted from being able to perform one or more of the CRUD operations on the resource based on a pre-configured access control policy. Dynamic authorization enables an M2M Device to be able to perform operations on resource(s) hosted at the M2M Gateway even when pre-configured access policies do not allow it. Dynamic authorization checks can be dictated based on dynamic authorization policies which can dictate the types of checks (e.g. higher-level of authentication checks, payment based authorization, platform validation checks etc.) that are performed. The authorization provided can be for a finite period of time and can be added to the static access control policy.

5.5.2 Actors

The entities involved in the use case are shown in Figure 5.5.2-1 and described as follows:

M2M Gateway: It represents a gateway that is responsible for hosting a resource. In addition, the M2M Gateway is responsible for implementing the Dynamic Authorization - Policy Enforcement (DA-PEP).

Trusted Third-Party: It represents an entity that is trusted both by an M2M Device as well by an M2M Gateway that is responsible for implementing the Dynamic Authorization - Policy Decision Point (DA-PDP) functionality as well as Dynamic Authorization - Policy Information Point (PIP). In addition it can implement or enable various types of authorization checks (e.g. multi-factor authentication, payment/subscription, platform integrity checks).

M2M Device: It represents a sensor application or a sensor device that is responsible for measuring sensor data and hosting a resource on an M2M Gateway.

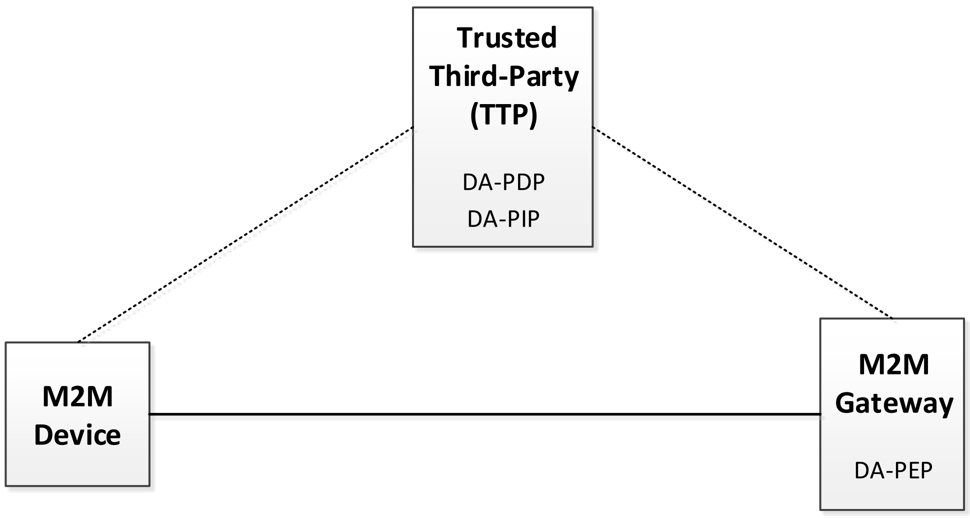


Figure 5.5.2-1: Entities involved in Dynamic Authorization

5.5.3 Pre-Conditions

The M2M Gateway and the Trusted Third-party are provisioned with appropriate dynamic authorization policies based on which, it decides on the dynamic authorization checks that will be performed.

It is assumed that all the communications that occur between the M2M Device, the M2M Gateways and the TTP occur over secure communications channel.

5.5.4 Normal Flow

Procedure for dynamic authorization:

1) An M2M Device requests to perform a CRUD operation on a resource hosted on the M2M Gateway.

2) The M2M Gateway determines that the M2M Device does not have the appropriate authorization based on a check on the pre-configured ACPs.

3) The M2M Gateway performs a request to the DA-PDP on the TTP in order to determine if the M2M Device can be granted access privileges via dynamic authorization.

4) The DA-PDP on the TTP determines that dynamic authorization can be carried out, then it determines the types of authorization checks that will be performed either with the M2M Device or on behalf of the device.

5) The DA-PDP on the TTP in conjunction with either the M2M Device or M2M Gateway performs one or more dynamic authorization checks (e.g. multi-factor authentication checks, platform validation checks, payment-based authorization, subscription, etc.) or enables the M2M Device to communicate with other authorization enabling functions in order to achieve the authorization.

6) If the authorization checks are successfully completed, then the DA-PDP on the TTP sends a response message containing authorization information (e.g. authorization level, validity of the authorization, operation/resources that can be performed by the M2M Device, etc.) to the DA-PEP on the M2M Gateway. The authorization information can be represented in varied forms (e.g. signed token(s), un-signed token(s), XML message, oneM2M messaging/resource structures, etc.). Additionally, in certain cases, the DA-PDP can optionally update the ACP to include the information about the M2M Device and the resource/operations that it is allowed to perform based on the dynamic authorization results.

7) The DA-PEP on the Gateway updates the relevant entries so that the M2M Device is able to perform the requested CRUD operations on resource(s).

8) The M2M Gateway provides a response to the M2M device indicating the authorization that was provided to it.

5.5.5 Potential Requirements

1) The M2M system shall support distributed dynamic authorization functions.

2) The M2M system shall support and enable various authorization checks (e.g. multi-factor authentication, platform validation, payment authorization etc.) in order to enable dynamic authorization.

3) The M2M system shall support various means to provide authorization information (e.g by means of signed token(s), un-signed token, XML messaging, JSON data, oneM2M messaging/resource structures).

4) The M2M system shall support an authorized entity (e.g. DA-PDP) to update the relevant ACPs based on dynamic authorization results

6 High Level Architecture

6.1 Review of existing frameworks

6.1.1 Introduction

Materials in this section are inspired from deliverables developed under European Projects SITAC and SUNSEED.

6.1.2 OAuth 2.0

OAuth 2.0 [i.6] is an authorization framework that aims to provide a secure way to allow third-party applications to get access to data on behalf of resource owners through a set of defined request flows. The main purpose of OAuth is to enable the user (resource owner, RO) to grant services and applications (client) access to protected resources stored at a resource server (RS), without sharing user credentials with the application. Instead of using the resource owner's credentials to access protected resources, the client gets an access token. Access tokens are issued to third-party clients by an authorization server (AS) after the approval of the issuance by the resource owner.

This protocol is widely used in the web today, primarily to allow users to register/log in into web platforms using their pre-existing credentials from well-known identity providers, but cannot be directly applied in M2M/IoT scenario. To make it suitable to oneM2M systems, the main issues to be addressed are:

- OAuth-defined flows assume that the user operates the third party service that the user is authorizing. In oneM2M scenarios, in contrast, the user operating the service that requests access to a resource might not be its owner.

- OAuth resource servers and scopes are defined at design time. However, in a distributed IoT architecture where services are deployed by end users at any time, a dynamic registration of resource servers relies on the ability to introduce new objects in the oneM2M model.

- The control of user privacy resides on the AS, which is a centralized point in the OAuth architecture. Although OAuth does not prevent deployment of multiple authorization servers, it is a topic not covered in the protocol specification, which raises many issues when using the standard "out-of-the-box" (e.g. authorization servers discovery, how to bind a resource server with a specific AS).

As discussed earlier, the resource owner is the source in all OAuth flows. The whole protocol is designed around that assumption. On the contrary, in IoT scenarios, users can be requesting access to resources that they do not own. As a consequence of that, OAuth alone cannot be used to support all oneM2M use cases.

Regarding the third issue described, it can be argued that OAuth also provides ways to deploy distributed authorization schemes. However, it is very focused on resource-owner driven scenario and most of the existing implementations are mainly focused on the register/log in use case. In spite of that, OAuth, because of is extensibility, can be used to accommodate additional scenarios.

6.1.3 User Managed Access (UMA)

User-Managed Access [i.7] is a profile of the OAuth 2.0 protocol that supports delegation of authorization in user oriented (rather than machine oriented) scenarios. It is intended to encompass the scenario where a resource owner wants to manage the access control to protected resources by clients operated by requesting parties, other than the resource owner. While the OAuth specification is focused on Alice-to-client sharing model, UMA focuses on Alice-to-Bob sharing model, where Alice can share proactively her resources before Bob even knows they exist. In addition, Bob can request a permission. In that case, a notification will be sent to Alice for her to decide whether she wants to grant access to Bob. Alice will collect these considerations from the AS.

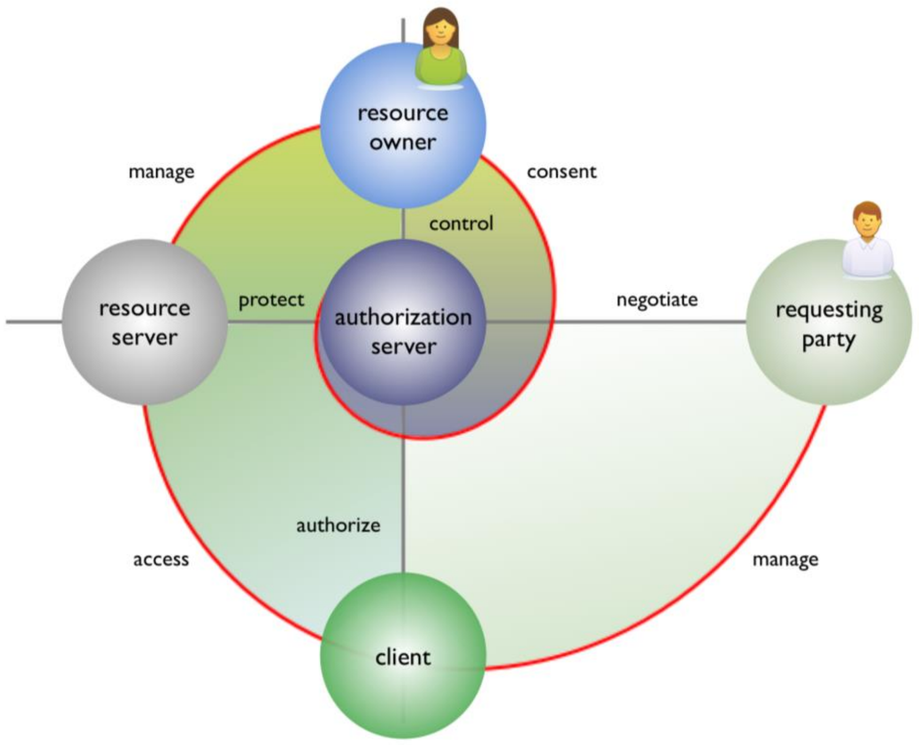


Figure 6.1.3-1: UMA actors and architecture

To achieve this, UMA defines new roles and interactions on top of OAuth as illustrated in Figure 6.1.3-1. Moreover, the roles defined in OAuth specification are slightly changed to accomplish UMA's objectives. From the UMA viewpoint, RS are clients for the AS, effectively decoupling RS and AS roles. In UMA, an RS can choose the AS where he wants to delegate the authorization. It is important to stress the point that no previous relationship between them is assumed. The other main difference is the introduction of the requesting party role. A requesting party is an entity that seeks access to a protected resource and can be different to the RO. The whole point of the introduction of this new role is to be able to distinguish between resource owner and resource consumer. Therefore, the RO can share his resources preserving the control over them, while any requesting party other than the resource owner can either request access to the resource or access it directly, provided that it has been granted access beforehand.

UMA defines the set of interactions that performed before a requesting party can access a resource. This interactions can be grouped in three phases: protecting a resource, getting access permission for a resource and accessing a resource. UMA is focused on access control management and does not specify data access protocols.

The capability to control access to resources at the action level is referred in this document as "fine grain authorization capability". Both figure 6.1.3-1 and figure 6.1.3.2 shows a resource server holding resources owned by a resource owner. Those resources can be accessed by a requesting party using a client application software, e.g. an oneM2M AE operating on its behalf. The administration of the access control to the resources can be cumbersome or difficult in the case where the resource owner owns multiple resources spread among multiple resource servers, because this administration task is performed in multiple places. This is where UMA can help, by enabling each resource server to delegate the administration to some or all of the resources it controls on behalf of the resource owner to a separate authorization server. This is done in a first phase by having the resource server register with the authorization server all resources the administration of which it wishes to delegate. In a second step, the client software acting on behalf of the requesting party will negotiate access to the resource with the authorization server. A successful negotiation will result in the client being provided with a Requesting Party Token (RPT), which when presented to the resource server will enable access to the protected resource.

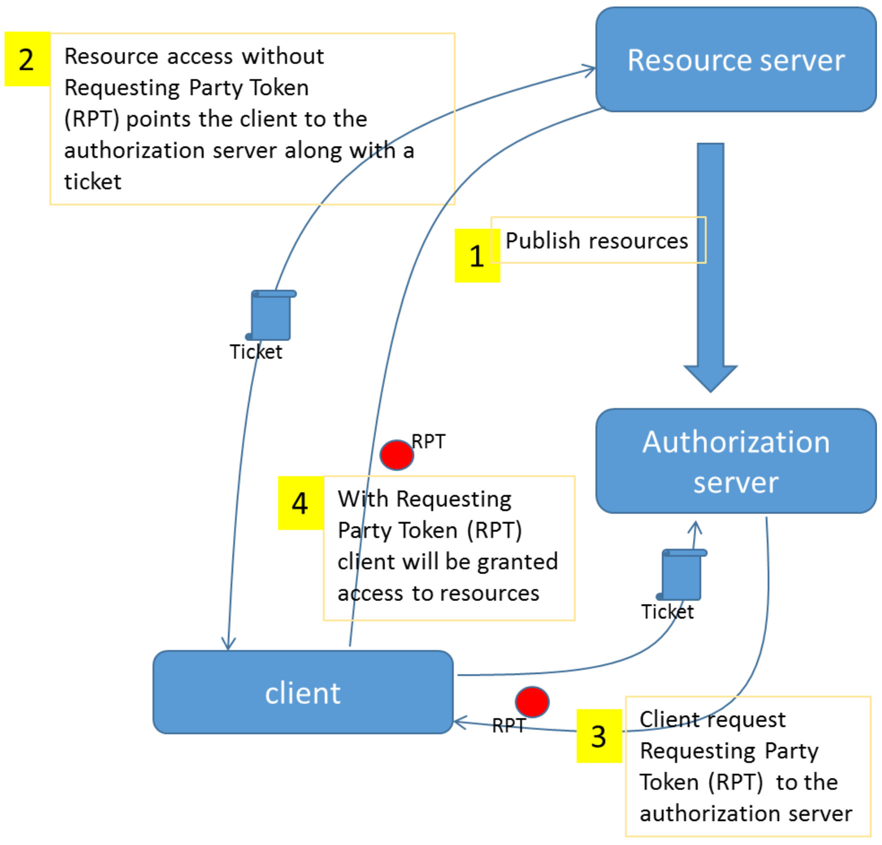

2
Resource access without Requesting Party Token (RPT) points the client to the authorization server along with a ticket
Resource server
1
Publish resources
Ticket
RPT
Authorization server
4
With Requesting Party Token (RPT) client will be granted access to resources
Ticket
client
RPT
3
Client request Requesting Party Token (RPT) to the authorization server

Figure 6.1.3-2: UMA APIs and flow of operations

Figure 6.1.3-2 shows the different APIs and flows of operation involved in UMA.

In a preliminary phase not shown on this figure the resource server and the authorization server have bootstrapped their relationship and as a result of this the resource server has obtained the possibility to invoke a resource publication REST API exposed by the authorization server for publishing the description of the resources it wishes to place under the protection of the authorization server. (step1).

In step 2, the client attempts to access the resource in the resource server without presenting a requesting party token. It is then pointed to the authorization server and given an "authorization ticket" indicating what is to be authorized. The list of actions to be authorized is described as an array of "scopes". Upon authentication and check of the authorization policies, the authorization server delivers a Requesting Party Token (RPT) to the client (step 3). Finally, the client is granted access to the resource upon presentation of this token to the resource server (step 4).

UMA offers many advantages for M2M/IoT scenarios, where users and organizations want to share their resources while keeping control both over the entities (friends, organizations, services, etc.) that have access to them and over the granularity of the information shared. This scenario can be modeled using UMA features, like resource registration and scope definition. Finally, due to the fact that the RS and the AS are decoupled in UMA, users can choose which entity they want to delegate the access control of his resources, instead of binding them to a particular provider.

In a dynamic and distributed environment, the access control architecture is expected to be flexible enough not only to accept new entities at any time, but also to modify their roles. In the oneM2M ecosystem, users will register new resources when they want to share information, and therefore the architecture designed is expected to allow new resource servers at runtime. While OAuth does not allow the introduction of new resource servers, UMA is able to handle this case.

6.1.4 OpenID Connect

OpenID Connect [i.8] represents another extension of OAuth 2.0 protocol, it is focused on the use case of sharing the end-user identity with clients. This use case is usually implemented through OAuth by giving applications an access token that allows them to get information of the user. It is often seen as an "Authorization over my identity" while, in essence, it represents an authentication process. OpenID Connect tries to address this use case and simplifies the protocol flow while preserving extensibility and compatibility with different identity providers.

As oneM2M supports managing different sources of user identities, OpenID Connect is a good candidate to accomplish this task. UMA can also be integrated with OpenID Connect. While UMA does not enforce the use of unique identifiers in the Authorization Server, unique identifiers are used for access control policies. In this case, the end-user will authorize the AS to query an identity endpoint for their unique (or non-unique) identity data.

6.2 Dynamic Authorization Architecture Proposal (DAA1) Reference Model

6.2.1 Overall Description

Figure 6.2.1-1 shows the oneM2M Dynamic Authorization Architecture Proposal 1 (DAA1) reference model. The DAA1 reference model is similar to that used for OAuth 2.0 [i.6].

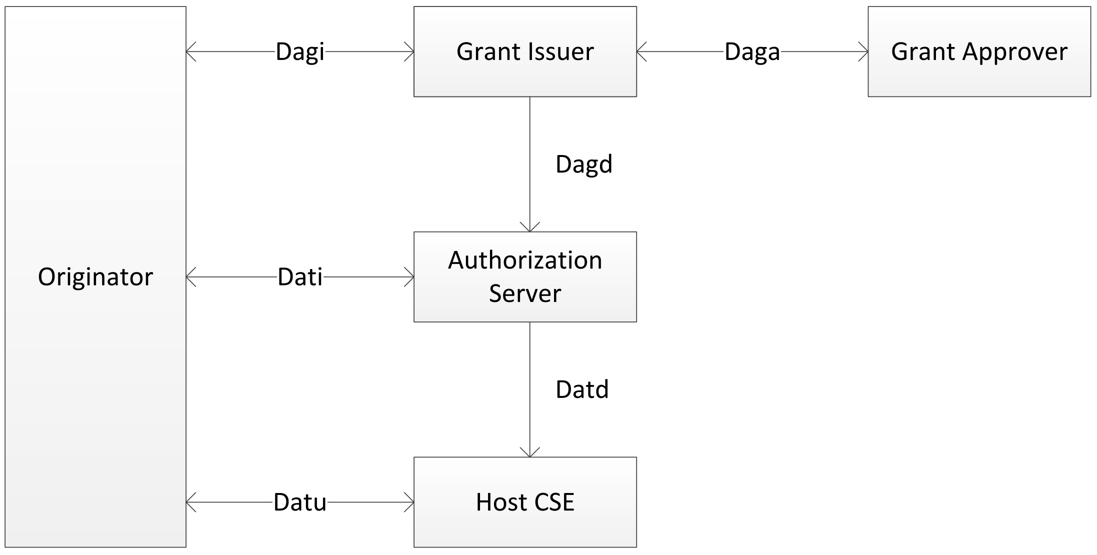


Figure 6.2.1-1: oneM2M Dynamic Authorization Architecture Proposal 1 (DAA1) Reference Model.

The DAA1 reference model and OAuth 2.0 both utilize the two types of dynamically-issued tokens:

- An *access token* represents an authorization for the Originator to access resources. The Originator presents the Host CSE with a request to access resources and access token(s). The Host CSE verifies the access token and then processes the request based on the authorization represented by the access token.

- An *authorization grant* describes an authorization to access resources and provides evidence that the Originator obtained authorization from the appropriate sources. There are two "classes" of the authorization grant. Some authorization grants are presented directly to the Host CSE as access tokens. Other authorization grants cannot be used in this way, and the Originator obtains a separate access token from the Authorization

Server; in this case, the authorization grant is presented to the Authorization Server to prove that the Originator has obtained authorization from the appropriate sources.

Some scenarios are better suited to issuing an authorization grant which is used directly as an access token. Other scenarios are better suited to issuing an authorization grant for which separate access tokens are obtained from an Authorization Server. Allowing both options provides flexibility.

An authorization grant is typically used is scenarios relying on the approval of the resource owner to enable access control. However in many M2M use case, the resource owners (i.e. the humans on behalf of which the M2M entities operate) cannot be expected to be easily reachable, whereas they would be able to configure the authorization server to make such decisions by itself.

Table 6.2.1-1 lists the functional roles in the DAA1 reference model. Clause 6.2.2 describes the functions associated with these functional roles. A oneM2M entity can assume multiple roles in this architecture model.

These functional roles generally correspond to functional roles defined for OAuth 2.0 [i.6]; the main difference is the partitioning of the OAuth 2.0 Resource Owner into a Grant Issuer CSE and Grant Approver, and additionally some names are changed (mostly to align with existing oneM2M terminology). The rationale behind partitioning the Resource Owner functionality is the following:

- The Authorization Server or the Host CSE (whichever processes the authorization grant) verifies the digital signature or MIC protecting the authorization grant. Consequently, the Authorization Server or the Host CSE is configured with credentials for verifying that digital signature or MIC.
- An architecture where the Grant Approver provides the digital signature or MIC (for the authorization grant) will rely on the Authorization Server or Host CSE managing credentials and identifiers for all possible Grant Approvers.
- A Grant Issuer can issue authorization grants for multiple Grant Approvers, and there are expected to be significantly fewer Grant Issuer than Grant Approvers.
- An architecture where the Grant Issuer provides the digital signature or MIC (for the authorization grant) will have significantly simplify credential management for the Authorization Server or Host CSE; when compared to an architecture where the Grant Approver provides the digital signature or MIC for the authorization grant.

The DAA1 reference model assumes the following sequence of events take place:

0) The Originator learns that an access token is a precondition to accessing some resources on the Host CSE.

1) Obtaining an authorization grant:

  a) The Originator sends a request to the Grant Issuer, asking to be issued an authorization grant which can be used by the Originator to access resources on the Host CSE.

  b) The Grant Issuer forwards the request to a Grant Approver that has sufficient permissions to approve issuing the requested authorization grant. The Grant Approver obtains a decision on issuing the authorization grant. For example, this process can include obtaining explicit approval from a person, or using an authorization architecture such as described in oneM2M TR-0016 [i.5]. The Grant Approver, then returns its decision (permitted or denied) to the Grant Issuer.

  c) On obtaining approval, the Grant Issuer forms an authorization grant, and returns this authorization grant to the Originator.

2) **Obtaining an access token (if applicable):** If the authorization grant cannot be presented to the Host CSE as an access token, then the Originator obtains an access token from the Authorization Server. The authorization grant is presented to the Authorization Server to prove that the Originator has obtained authorization from the appropriate source(s). Typically, an access token's lifetime is shorter than that of an authorization grant, and each authorization grant is re-used to obtain a series of access tokens (until the authorization grant expires).

3) **Accessing a resource using an Access Token**. The Originator sends the access token with the request sent to the Host CSE. The Host CSE verifies the access token and (for the purposes of processing the request) uses the permissions represented by the access token. Each access token is typically reused with multiple requests (until the access token expires).

Table 6.2.1-1: List of DAA1 functional roles

DAA1 Functional Role	Corresponding OAuth 2.0 Role [i.6]	Description	Details in clause
Originator	Client	See oneM2M TS-0001 [i.2]. Interacts with Grant Issuer and (Optionally) authorization servers to obtain authorization to access resources on the Host CSE.	6.2.2.1
Host CSE	Resource Server	See oneM2M TS-0001 [i.2]. Accepts authorizations issued by Authorization Server.	6.2.2.2
Grant Issuer	Resource Owner	An intermediary between the Originator and Grant Approver. This entity is recognized by the authorization server and Host CSE as having permission to issue authorization grants on behalf of Grant Approvers.	6.2.2.3
Grant Approver		This entity has permission to approve or deny issuing an authorization grant. For a given resource, there can be multiple entities permitted to be the Grant Approver for that resource.	6.2.2.4
Authorization Server	Authorization Server	This entity issues access tokens to Originators presenting valid authorization grants.	6.2.2.5

Table 6.2.1-2 lists the DAA1 reference points. Where a oneM2M entity assumes multiple functional roles, the reference points can be internal to that entity. Clause 6.2.3 describes the reference points in the DAA1 reference model.

Table 6.2.1-2: List of DAA1 reference points

Reference Point Identifier	Reference Point Descriptive Name	End Points	Description	Details in clause
Dagi	DAA1 Authorization Grant Issuance	Originator, Grant Issuer	Used for requesting and issuing authorization grants	6.2.3.1
Daga	DAA1 Authorization Grant Approval	Grant Issuer, Grant Approver	Used for obtaining approval to issue authorization grants	6.2.3.2
Dati	DAA1 Access Token Issuance	Originator, Authorization Server	Used for requesting and issuing access tokens	6.2.3.3
Datu	DAA1 Access Token Usage	Originator, Host CSE	Providing an access token as authorization to act on one or more resources	6.2.3.4
Dagd	DAA1 Authorization Grant Data	Grant Issuer, Authorization Server	Defines how authorization grants are formed at the Grant Issuer and processed at the Authorization Server. This data can traverse the Dagi and Dati reference points.	6.2.3.5
Datd	DAA1 Access Token Data	Authorization Server, Host CSE	Defines how access tokens are formed at the Authorization Server and processed at the Host CSE. This data can traverse the Dati and Datu reference points.	6.2.3.6

6.2.2 DAA1 Functional Roles

6.2.2.1 Originator

An Originator performs the following functions within the scope of DAA:

- Interact with a Grant Issuer to request and receive an authorization grant.
- Interact with an Authorization Server to present an authorization grant and receive an access token in return.

- Interact with a Host CSE to present an access token and receive access to one or more resources at the Host CSE.

6.2.2.2 Grant Issuer

A Grant Issuer performs the following functions within the scope of DAA:

- Receive, from an Originator, a request to issue an authorization grant.
- Verify that the identified Grant Approver has permission to approve authorization grants with the requested scope.
- Interact with a Grant Approver to receive a decision (allow/deny) regarding issuing an authorization grant.
- Issue an approved authorization grant.
- Send the authorization grant to the Originator.

6.2.2.3 Grant Approver

A Grant Approver performs the following functions within the scope of DAA:

- Receive, from the Grant Issuer, a request for approval to issue an authorization grant.
- Obtain a decision (allow or deny) regarding issuing an authorization grant. The details of this function is not specified in this document.
- Send the decision (allow or deny) to the Grant Issuer.

Examples of entities assuming the functional role of Grant Approver can include:

- A oneM2M AE on a user device (such as a smartphone, tablet or laptop), where the user device provides a user interface for approving authorization grants.
- A web server through which the user can approve authorization grants - with the user interface (for approving authorization grants) provided by a web-page or other application on a user device.
- A web server configured with policies for making automated decisions - much like a Policy Decision Point (PDP) in the authorization architecture proposed in oneM2M TR-0016 [i.5].

6.2.2.4 Authorization Server

An Authorization Server performs the following functions within the scope of DAA:

- Receive, from an Originator, a request to issue an access token which includes an authorization grant.
- Verify that the authorization grant is valid.
- Issue the requested access token, if permitted by the authorization grant.
- Send the access token to the Originator.

The Authorization Server is expected to be configured with policies allowing it to determine the validity of the authorization grant - in particular:

- The Authorization Server might be configured with credentials to verify the authenticity of the authorization grant (that is, verify that the Grant Issuer genuinely issued this authorization grant).
- The Authorization Server is configured with the range of permissions on the Host CSE for which the Grant Issuer is allowed to issue authorization grants.
- (If this feature is desirable) The Authorization Server is configured with the range of permissions on the Host CSE for which the Grant Approver is allowed to approve issue authorization grants.

6.2.2.5 Host CSE

A Host CSE performs the following functions within the scope of DAA:

- Receive, from an Originator, a request to perform actions on one or more resources, accompanied by an access token.
- Verify that the access token is valid.
- Perform the requested actions, if permitted by the access token.
- Send the access token to the Originator.

The Host CSE is expected to be configured with policies allowing it to determine the validity of the access token, in particular:

- The Host CSE might be configured with credentials to verify the authenticity of the access token (that is, the Host CSE might verify that the Grant Issuer or Authorization Server genuinely issued this access token).
- The Host CSE is configured with the range of permissions on the Host CSE for which the access token issuer (Grant Issuer or Authorization Server) is allowed to issue authorization grants.
- (If this feature is desirable) The Host CSE is configured with the range of permissions on the Host CSE for which the Grant Approver is allowed to approve issue authorization grants.

6.2.3 DAA1 Reference Points

6.2.3.1 DAA1 Authorization Grant Issuance (Dagi) Reference Point

This reference point is between the Originator and the Grant Issuer.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Originator to request an authorization grant from the Grant Issuer. This can include:
  - Enabling the Grant Issuer to verify which Originator sent the request (for example, using a digital signature or MIC).
  - Enabling the Originator to identify a Grant Approver to approve the request.
  - Enabling the Originator to identify the applicable Host CSE(s).
  - Enabling the Originator to indicate the scope of permissions to be granted by the authorization grant. Clause 7 will discuss how this scope is indicated.

  EXAMPLE: The scope can be a set of resources or a "role" used in an accessControlPolicy resource.

  - Enabling the Originator to identify the Authorization Server which will issue access tokens.
  - A desired time window for the authorization grant.
- Enabling the Grant Issuer to send a respond to the Originator ,which can comprise:
  - providing an error message; or
  - providing an authorization grant or a unique identifier for the authorization grant (which can be subsequently used by the Authorization Server to retrieve the authorization grant). The Grant Issuer can also provide additional information that the Originator might utilize regarding the authorization grant.
- Enabling the Originator and Grant Issuer to establish a credential that the Originator will use to prove to the Authentication Server that the authorization grant was issued to the Originator. This can impact the request and/or response across this reference point.

6.2.3.2 DAA1 Authorization Grant Approval (Daga) Reference Point

This reference point is between the Grant Issuer and the Grant Approver.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Grant Issuer to request approval, from the Grant Approver, to issue an authorization grant on behalf of the Grant Approver. This can include:
  - Providing details of the Originator's request for an authorization grant (see clause 6.2.3.1 "DAA1 Authorization Grant Issuance (Dagi) Reference Point").
  - Suggesting a different time window for the authorization grant.
  - Suggesting an authorization grant type.
- Enabling the Grant Approver to send a respond to the Grant Issuer which can comprise:
  - Decision (approved/denied).
  - Selected time window.
  - Selected authorization type.

The Grant Approver can be a oneM2M entity (CSE or AE), but the architecture reference model also allows the Grant Approver to be a non-oneM2M entity, provided it can communicate with the Grant Issuer. Consequently, while this reference point can involve communication between two oneM2M entities over Mcc or Mca, it is also possible that this reference point can involve communication over non-oneM2M reference points.

6.2.3.3 DAA1 Access Token Issuance (Dati) Reference Point

This reference point is between the Originator and the Authorization Server.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Originator to request an access token from the Authorization Server. This can include:
  - Providing an authorization grant or a unique identifier for the authorization grant (which can be subsequently used by the Authorization Server to retrieve the authorization grant).
  - Enabling the Authorization Server to verify that the Originator which sent the request is also the Originator to whom the authorization grant was issued (for example, using a digital signature or MIC).
  - Enabling the Originator to indicate the scope of permissions to be granted by the access token (if smaller than the scope of permissions represented by the authorization token). Clause 7 will discuss how this scope is indicated.

EXAMPLE: The scope can be a set of resources or a "role" used in an accessControlPolicy resource.

- Enabling the Authorization Server to send a respond to the Originator ,which can comprise:
  - providing an error message; or
  - providing an access token or a unique identifier for the access token (which can be subsequently used by the Host CSE to retrieve the access token). The Authorization Server can also provide additional information that the Originator might utilize regarding the access token.
- Enabling the Originator and Authorization Server to establish a credential that the Originator will use to prove to the Host CSE that the authorization grant was issued to the Originator. This can impact the request and/or response across this reference point.

6.2.3.4 DAA1 Access Token Usage (Datu) Reference Point

This reference point is between the Originator and the Host CSE.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Originator to provide an access token with a request to act on a resource(s) to the Host CSE. This can include:
  - Providing an access token or a unique identifier for the access token (which can be subsequently used by the Host CSE to retrieve the access token).
  - Enabling the Host CSE to verify that the Originator which sent the request to act on resource(s) is also the Originator to whom the access token was issued (for example, using a digital signature or MIC).
- Enabling the Host CSE to provide an error message to the Originator, which can comprise:
  - Identifying that the access token is no longer valid.
  - Identifying an authorization server that the Host CSE allows to issue access tokens. This is used when either no access token was provided, or an access token was provided but not recognized by the Host CSE.

6.2.3.5 DAA1 Authorization Grant Data (Dagd) Reference Point

This reference point is between the Grant Issuer and the Authorization Server.

This reference point defines how authorization grants are formed at the Grant Issuer and processed at the Authorization Server.

The authorization grant data can traverse the Daga and Dati reference points via the Originator.

Alternatively, the authorization grant data can be communicated directly from the Grant Issuer to the Authorization Server, with the Daga and Dati reference points communicating a unique identifier for the authorization grant (rather than communicating the authorization grant data itself). This alternative can suit scenarios where the roles of Grant Issuer and Authorization Server are assumed by a single entity.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Identify the Originator authorized by the authorization grant.
- Identify the Grant Issuer that issued the authorization grant.
- Verify that the identified Grant Issuer issued the authorization grant.
- Identify the Grant Approver that approved issuing the authorization grant.
- Identify the Authorization Server for which the authorization grant is intended.
- Identify the Host CSE(s) for which resulting access token(s) are intended.
- Identify the permissions represented by the authorization grant.
- Identify the time window within which the authorization grant is valid.

6.2.3.6 DAA1 Access Token Data (Datd) Reference Point

This reference point is between the Authorization Server and the Host CSE.

This reference point defines how access tokens are formed at the Authorization Server and processed at the Host CSE.

The access token data can traverse the Dati and Datu reference points via the Originator.

Alternatively, the access token data can be communicated directly from the Authorization Server to the Host CSE, with the Dati and Datu reference points communicating a unique identifier for the access token (rather than communicating the access token data itself). This alternative can suit scenarios where the roles of Authorization Server and Host CSE are assumed by a single CSE.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Identify the Originator authorized by the access token.
- Identify the Authorization Server that issued the access token.
- Verify that the identified Authorization Server issued the access token.
- Identify the authorization grant used as the basis for issuing this access token.
- Identify the Grant Issuer that issued the authorization grant.
- Identify the Grant Approver that approved issuing the authorization grant.
- Identify the Host CSE(s) for which the access token is intended.
- Identify the permissions represented by the access token.
- Identify the time window within which the access token is valid.

6.3 Dynamic Authorization Architecture Proposal 2 (DAA2) Reference Model

6.3.1 Access Token Issuance Architecture

Figure 6.3.1-1 provides a high level overview of a generic token issuance architecture. This architecture includes the following entities.

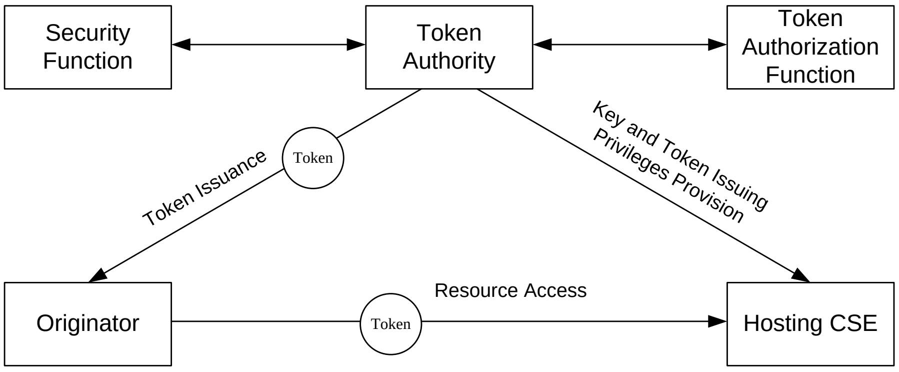


Figure 6.3.1-1: DAA2 Token issuance architecture

Table 6.3.1-1 lists the functional roles in the DAA2 reference model. An oneM2M entity can assume multiple roles in this architecture model.

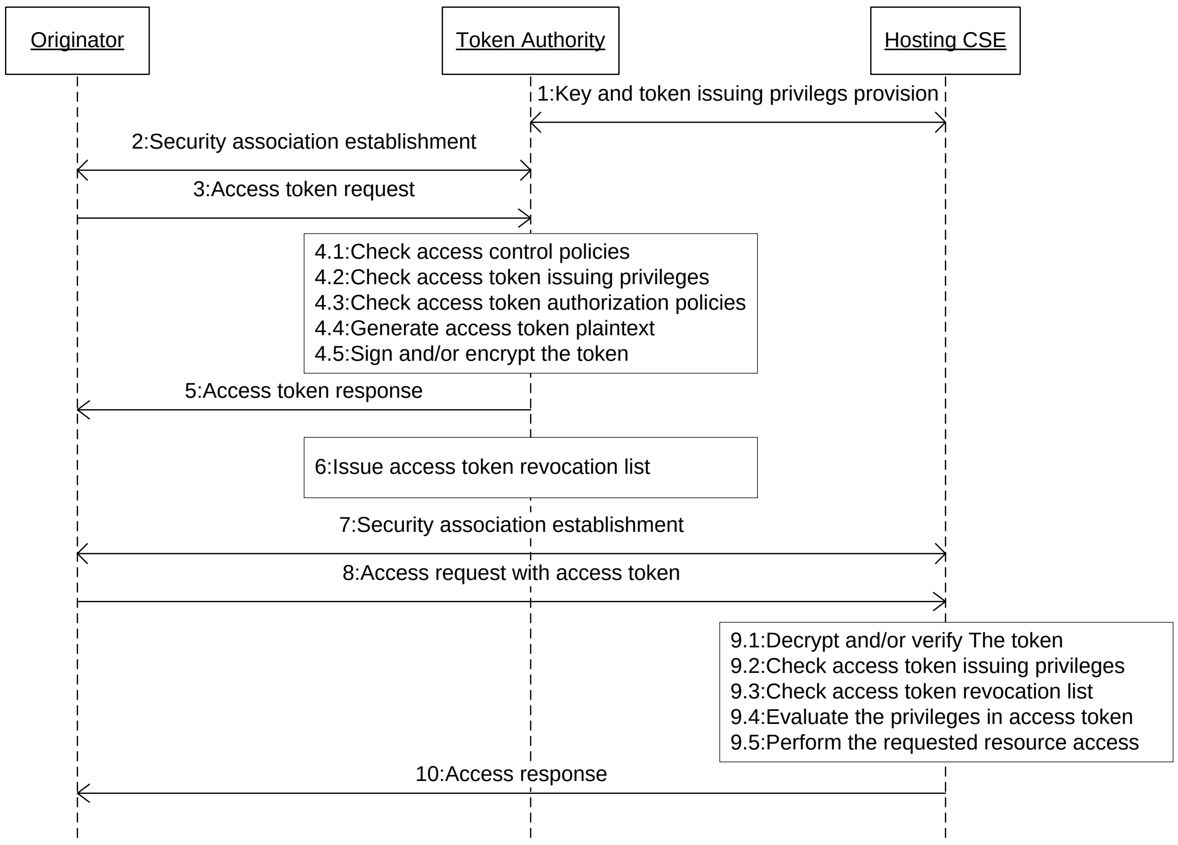
Table 6.3.1-1: List of DAA2 Functional Roles

DAA2 Functional Role	Corresponding OAuth 2.0 Role [i.6]	Description
Originator	Client	See oneM2M TS-0001 [i.2]. This entity is the holder of an Access Token. It uses Access Tokens to get permission for accessing resources.
Token Authority	Authorization Server	This entity is responsible for issuing Access Tokens.
Host CSE	Resource Server	See oneM2M TS-0001 [i.2]. This entity provides the services of accessing resources according to privileges described in Access Tokens.

The Token Authority can contact a Token Authorization Function to determine or confirm the privileges to be included in an Access Token. The Token Authority can contact a Security Function to generate signed and/or encrypted Access Tokens. The present document does not describe the functional roles of Token Authorization Function and Security Function, and does not describe the interactions with these entities.

6.3.2 General Procedure of Token Issuance and Use in DAA2

The generic procedure of token issuance and use in DAA2 is shown in figure 6.3.2-1 and described as follows.



Originator
Token Authority
Hosting CSE
1:Key and token issuing privilegs provision
2:Security association establishment
3:Access token request
4.1:Check access control policies
4.2:Check access token issuing privileges
4.3:Check access token authorization policies
4.4:Generate access token plaintext
4.5:Sign and/or encrypt the token
5:Access token response
6:Issue access token revocation list
7:Security association establishment
8:Access request with access token
9.1:Decrypt and/or verify The token
9.2:Check access token issuing privileges
9.3:Check access token revocation list
9.4:Evaluate the privileges in access token
9.5:Perform the requested resource access
10:Access response

Figure 6.3.2-1: Generic process of Access Token issuance and use in DAA2

1) The Token Authority and the Hosting CSE are pre-configured with the keys used for generating or verifying access tokens and Access Token Issuing Privileges of the Token Authority.

2) The Originator and the Token Authority establish security association through mutual authentication to ensure the integrity and confidentiality of communications between the two entities.

3) The Originator sends Access Token Request to the Token Authority.

4) The Token Authority CSE carries out the following operations:

    1) Check if the Originator has the privileges of accessing the target resource according to the ACP.

    2) Check if the requested privileges are within the Access Token Issuing Privileges of the Token Authority. This operation can be performed by a Token Authorization Function on behalf of the Token Authority.

    3) Check if the Originator can obtain the privileges described in the Access Token Request according to the Access Token Authorization Policies. This operation can be performed by a Token Authorization Function on behalf of the Token Authority.

    4) Generate access token plaintext.

    5) Sign and/or encrypt the access token plaintext.

5) The Token Authority returns the issued Access Token back to the Originator.

6) It is also possible for the Token Authority to store the Access Token and inform the Originator where to retrieve it.

7) The Token Authority can also issue an Access Token Revocation List in which the revoked Access Tokens that are still not expired are placed.

8) The Originator and the Hosting CSE establish security association through mutual authentication to ensure the integrity and confidentiality of communications between the two entities.

9) The Originator sends a resource access request to the Hosting CSE.in which one or multiple Access Tokens are included

10) The Hosting CSE extracts the Access Tokens from the resource access request, and carries out the following operations:

    1) Decrypt and/or verify the Access Token.

    2) Check if the privileges in the Access Token are within the Access Token Issuing Privileges of the Token Authority.

    3) Check if this Access Token has been revoked against an Access Token Revocation List.

    4) Evaluate the Originator's resource access request based on the privileges in the Access Token.

    5) Perform the requested resource access.

11) The Hosting CSE returns the execution result back to the Originator.

6.4 Dynamic Authorization Architecture Proposal 3 (DAA3) Reference Model

6.4.1 Overall Description

Figure 6.4.1-1 shows the oneM2M Dynamic Authorization Architecture proposal 3 (DAA3) reference model. The DAA3 reference model is similar to that used for OAuth 2.0 [i.6], borrowing some ideas from UMA [i.7]. The figure shows logical reference points; communication paths for a reference point can pass through other entities shown in the diagram, and through additional entities not shown in the figure. In the context of DAA3, and access token is analogous to a UMA requesting party token (RPT).

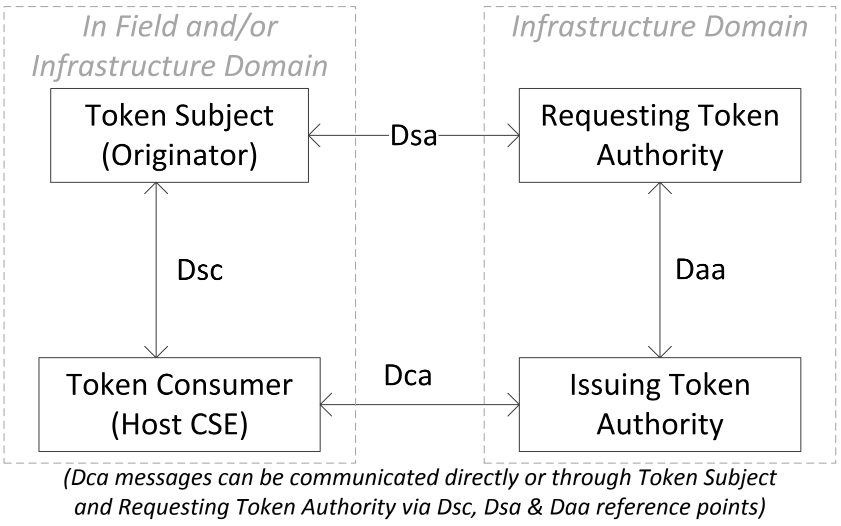

In Field and/or Infrastructure Domain
Infrastructure Domain
Token Subject (Originator)
Dsa
Requesting Token Authority
Dsc
Daa
Token Consumer (Host CSE)
Dca
Issuing Token Authority
(Dca messages can be communicated directly or through Token Subject and Requesting Token Authority via Dsc, Dsa & Daa reference points)

Figure 6.4.1-1: oneM2M Dynamic Authorization Architecture Proposal 3 (DAA3) Reference Model.

Table 6.4.1-1 lists the functional roles in the DAA3 reference model. Clause 6.4.2 describes the functions associated with these functional roles.

Table 6.4.1-1: List of DAA3 functional roles

DAA3 Functional Role	Corresponding OAuth 2.0 Role [i.6]	DAA3 Sub-Roles	Description	Details in clause
Token Subject	Client	-	See oneM2M TS-0001 [i.2]. The entity being provided with authorization to access resources on the Token Consumer. The Token Subject's Token Authority obtains access tokens for this entity	6.4.2.1
Token Authority	Authorization Server	Requesting Token Authority	Obtains access tokens on behalf of the Token Subject. This entity interacts with the Issuing Token Authority to provide information about the Token Subject.	6.4.2.2.1
		Issuing Token Authority	This entity issues access tokens, to Token Subjects, which are trusted by the Token Consumer. This entity interacts with the Requesting Token Authority to obtain information about the Token Subject for use in deciding whether to issue an access token.	6.4.2.2.2
Token Consumer	Resource Server	-	See oneM2M TS-0001 [i.2]. Accepts authorizations issued by the Issuing Token Authority.	6.4.2.3

These functional roles generally correspond to functional roles defined for OAuth 2.0 [i.6] and UMA [i.7]:

- The role of the OAuth 2.0/UMA client is called the *Token Subject* in DAA3 - this corresponds to the oneM2M Originator with respect to the resource access request for which the access token provides authorization.

- The role of the OAuth 2.0/UMA resource server is called the *Token Consumer* in DAA3 - this corresponds to the oneM2M Host CSE with respect to the resource access request for which the access token provides authorization.

- The role of the OAuth 2.0/UMA resource owner corresponds to a stakeholder and not a functional entity. The interactions with this stakeholder can be proprietary and would be managed by the Token Authority in this reference model. Further, the nature of "ownership" is very complex in M2M use cases - so this term has the potential cause confusion. Consequently, the DAA3 reference model does not include a functional role equivalent to the OAuth 2.0/UMA resource owner.

- The role of the OAuth 2.0/UMA authorization server is called *Token Authority* in DAA3 - primarily to clarify that this entity provides authorization by providing the Token Subject with an access token (as differentiated from services described in oneM2M TR-0016 [i.5]). The Token Authority is assumed to reside in the Infrastructure Domain. A Token Authority can assume one or both of the following roles with respect to the Token Consumer and Token Subject:

- If a Token Consumer accepts tokens issued by the Token Authority, then the Token Authority can assume the role of *Issuing Token Authority* with respect to the Token Consumer. The DAA3 reference model assumes that a trust relationship exists between the Token Consumer and the Issuing Token Authority.
- The Issuing Token Authority makes the decision to issue a token based on policy and trustworthy information about the Token Subject client and/or stakeholder on whose behalf the Token Subject is requesting access (in UMA this stakeholder is called the requesting party). In cases where the Issuing Token Authority does not have direct access to the trustworthy information about the Token Subject and/or UMA requesting party, then the Issuing Token Authority is provided this information by a trusted third party. In UMA [i.7], the OAuth 2.0 authorization server can request information via Token Subject, and the Token Subject to present "claims" containing this information. However, this interaction - along with the interactions necessary to obtain the "claims" introduces multiple round trips of communication involving the Token Subject. If the Token Subject is in the Field Domain, then this introduces a large overhead in terms of time and communication. In M2M scenarios, it is much more efficient for an entity in the infrastructure domain to act as a proxy for the Token Subject, interacting with the Issuing Token Authority to request an access token on behalf of the Token Subject and provide the information requested by the Issuing Token Authority. The DAA3 reference model assumes that entity assuming this functionality is also a Token Authority. The role of this entity in DAA3 is the *Requesting Token Authority*. There is no corresponding functional role in OAuth 2.0 or UMA. The DAA3 reference model assumes that:

    - a trust relationship exists between the Token Subject and the Requesting Token Authority; and
    - a trust relationship exists between the Requesting Token Authority and the Issuing Token Authority.

  The DAA3 reference model allows a Token Authority to assume the roles of both the Issuing Token Authority and Requesting Token Authority. In particular, if a trust relationship exists between the Token Subject and the Issuing Token Authority, then the Issuing Token Authority can also assume the role of Requesting Token Authority.

There are two main options for securing the interactions between Token Consumer and Issuing Token Authority (over Dca):

- The Token Consumer is "online" if the Token Consumer can establish a secure association (e.g. TLS or DTLS) with the Issuing Token Authority. In this case, the Token Consumer and Issuing Authority would exchange the authorization data details and access token details directly (secured by TLS or DTLS), and the only data passed via the Token Subject are "pointers" to the authorization data details and access token details.
- The Token Consumer is "offline" if the Token Consumer cannot establish a secure association with the Issuing Token Authority, and instead end-to-end security is used. In this case, the Token Consumer and Issuing Authority can exchange the authorization data details and access token details (secured end-to-end) either directly or via the Token Subject.

Similarly, there are two main options for securing the interactions between Token subject and Requesting Token Authority (over Dsa):

- The Token Subject is "online" if the Token Subject can establish a secure association (e.g. TLS or DTLS) with the Requesting Token Authority.
- The Token Subject is "offline" if the Token Subject cannot establish a secure association with the Requesting Token Authority, and instead end-to-end security is used.

The DAA3 reference model assumes the following sequence of events take place (a detailed message flow is provided in clause 6.4.4) - alternative flows are possible, and are considered in clause 8.

0) **Setup** (These procedures are not addressed in the present document):

- The Issuing Token Authority is provided with policies governing issuing access tokens covering authorization data provided by the Token Consumer. The policy descriptions are not addressed in the present document.
- The Requesting Token Authority is provided with information about the Token Subject which can be passed to Token Authorities for making token-issuing decisions.
- The Token Subject is configured with the identity of the Requesting Token Authority which interacts with the Issuing Token Authority, on behalf of the Token subject, to obtain access tokens. The Requesting Token Authority and the Token Subject have credentials for securing communication between those entities (that is, the Dca reference point). These credentials can be used for security association establishment (providing "online" security) or for end-to-end security (providing "offline" security). The Token Subject can be configured for multiple Requesting Token Authorities, which are contacted to see if they are able to interact with the Issuing Token Authority on behalf of the Token Subject.
- The Issuing Token Authority and the Token Consumer are pre-configured with the Access Token Issuing Privileges of the Issuing Token Authority. The Issuing Token Authority and the Token Consumer are pre-configured with the credentials for securing communication between those entities (that is, the Dca reference point). These credentials can be used for security association establishment (providing "online" security) or for end-to-end security (providing "offline" security).

NOTE: The subsequent steps can depend on whether the Token Consumer is online or offline, and whether the Token Subject is online or offline and whether the Token Subject communicates with the Request Token Authority via the Token Consumer or independent from the Token Consumer. The current description assumes that the Token Consumer is offline and that the Token Subject communicates with the Request Token Authority independent from the Token Consumer.

1) **Initialization (Optional - if the Token Subject already knows submitTo and authData, then these steps can be skipped):** The Token Subject submits a resource access request for which the Token Consumer determines that the Token Subject does not have sufficient permissions. In response, the Token Consumer:

   a) Denies the Token Subject's request to access resources.

   b) Identifies where access token requests can be submitted to the appropriate Issuing Token Authority.

   c) Provides the *authorization data* to be used by the Issuing Token Authority in composing the access token. The authorization data describes the privileges to be provided to the Token Subject for its resource access request- and which the Issuing Token Authority would provide when it issues the corresponding access token (subject to the Token Subject meeting the conditions of the appropriate policies).

   This interaction occurs over the Dsc reference point.

NOTE: Initialization is not addressed in the core OAuth 2.0 specification [i.6]. UMA provides a process similar to Initialization - in UMA, the authorization data is communicated from the Token Consumer (OAuth 2.0/UMA resource server) directly to the Issuing Token Authority (OAuth 2.0/UMA Authorization Server), and the Token Consumer provides the Token Subject (OAuth 2.0/UMA Client) with corresponding "permission ticket" which is then provided to the Issuing Token Authority (rather than communicating the authorization data via the Token Subject). This assumes that the Token Consumer can easily communicate with the Issuing Token Authority -which is not always the case for Token Consumer in the Field Domain. Consequently, the general message flow for DAA3 in clause 6.4.4 "Example Procedure of Access Token Issuance and Use in DAA3" assumes the authorization data is communicated via the Token Subject. Clause 8 can explore the use of "permission tickets" and other optimizations.

2) **Obtaining an Access Token:** The Token Subject forwards the returned information to a Requesting Token Authority. If the Requesting Token Authority approves obtaining a token, then the Requesting Token Authority provides the Issuing Token Authority with the forwarded authorization data and any additional applicable information about the Token Subject used by the Issuing Token Authority's decision process. The Issuing Token Authority applies policies to decide whether to issue an access token. For example, this process can include obtaining explicit approval from a person, or using an authorization architecture such as described in oneM2M TR-0016 [i.5]. The Issuing Token Authority generates an access token, and returns this to the Requesting Token Authority. The Requesting Token Authority forwards the access token to the Token Subject. This interaction occurs over the Dsa and Daa reference points.

3) **Accessing a resource using an Access Token:** The Token Subject sends the access token with the request sent to the Token Consumer. The Token Consumer verifies the access token and (for the purposes of processing the request) uses the permissions represented by the access token. Each access token is typically reused with multiple requests (until the access token expires). This interaction occurs over the Dsc reference point.

Table 6.4.1-2 lists the DAA3 reference points. Where an oneM2M entity assumes multiple functional roles, the reference points can be internal to that entity. Clause 6.4.3 describes the reference points in the DAA3 reference model.

Table 6.4.1-2: List of DAA3 reference points

Reference Point Identifier	Reference Point Descriptive Name	End Points	Description	Details in clause
Dsa	DAA3 Subject-to-Authority	Token Subject, Requesting Token Authority	Requesting the Requesting Token Authority to obtain an access token on behalf of the Token Subject.	6.4.3.1
Daa	DAA3 Authority-to-Authority	Requesting Token Authority, Issuing Token Authority	Used for requesting and issuing access tokens.	6.4.3.2
Dsc	DAA3 Subject-to-Consumer	Token Subject, Token Consumer	Serves two purposes: 1) the Token Consumer providing the Token Subject with information for obtaining an access token; and 2) the Token Subject providing the Token Consumer with an access token as authorization to act on one or more resources.	6.4.3.3
Dca	DAA3 Consumer-to-Authority	Token Consumer, Issuing Token Authority	Defines how the Token Consumer and Issuing Token Authority exchange authorization data and access tokens. This data traverses the remaining reference points (Dsc, Dsa, Daa).	6.4.3.4

6.4.2 DAA3 Functional Roles

6.4.2.1 Token Subject

A Token Subject performs the following functions within the scope of DAA3:

- Receive information from the Token Consumer enabling the Token Subject to request an access token suitable for requested resource access.
- Requesting the Requesting Token Authority to obtain an access token on behalf of the Token Subject. Where the Token Subject is configured with the addresses of multiple Requesting Token Authorities, the Token Subject asks Requesting Token Authorities until a Requesting Token Authority is found which can obtain an access token on behalf of the Token Subject.
- Interact with a Token Consumer to present an access token and receive access to one or more resources at the Token Consumer.

Assumptions regarding the Token Subject:

- The Token Subject is configured with the addresses of one or more Requesting Token Authorities which, on behalf of the Token Subject, might be able to interact with the Issuing Token Authority.

6.4.2.2 Token Authority

6.4.2.2.1 Introduction

Within the scope of DAA3, a Token Authority is able to perform the functions of:

- a Requesting Token Authority, described in clause 6.4.2.2.1 "Requesting Token Authority"; and
- an Issuing Token Authority, described in clause 6.4.2.2.2 "Issuing Token Authority".

6.4.2.2.2 Requesting Token Authority

A Requesting Token Authority performs the following functions within the scope of DAA3:

- Receive, from the Token Subject, a request to obtain an access token which includes authorization data and identifies where the access token request is to be submitted.
- Apply policies to decide if the Requesting Token Authority submits a token request to the Issuing Token Authority on behalf of the Token Subject.
- Any obligations on the part of the Requesting Token Authority (for example, obtaining explicit approval from a user or administrator for individual token requests) are not detailed in the present document.
- Submit a request to the identified Issuing Token Authority, including the authorization data.
- Provide necessary information about the Token Subject to the Issuing Token Authority.
- Receive an access token from the Issuing Token Authority.
- Forward the access token to the Token Subject.

Assumptions regarding the Requesting Token Authority:

- The Requesting Token Authority is assumed to be configured to interact with the Issuing Token Authority on behalf of the Token Subject.
- The Requesting Token Authority is expected to be configured with policies regarding decisions to request access tokens.

Examples of entities that can assume this functional role can include:

- A web server through which the user can approve authorization grants - with the user interface (for approving authorization grants) provided by a web-page or other application on a user device.
- A web server configured with policies for making automated decisions if the Requesting Token Authority submits a token request to the Issuing Token Authority on behalf of the Token Subject - much like a Policy Decision Point (PDP) in the authorization architecture proposed in oneM2M TR-0016 [i.5].

6.4.2.2.3 Issuing Token Authority

A Issuing Token Authority performs the following functions within the scope of DAA3:

- Receive, from a Requesting Token Authority, a request on behalf of the Token Subject to issue an access token which includes authorization data.
- Apply end-to-end security processes to the authorization data in the access token request if applicable (e.g. decryption, signature verification).
- Determine applicable policies for the authorization data, including determining what information about the Token Subject is applicable.
- Obtain necessary information about the Token Subject from the Requesting Token Authority which submitted the access token request on behalf of the Token Subject.

- Any obligations on the part of the Issuing Token Authority (for example, obtaining explicit approval from a user or administrator for individual token requests) are not detailed in the present document.
- Make decisions whether to issue an access token providing the privileges described in the authorization data.
- Issue the requested access tokens.
- (Optionally) Apply end-to-end security processes to the access token (e.g. encryption signature generation), with the corresponding security processing to be applied by the Token Consumer.
- Send the access token to the Requesting Token Authority.

Assumptions regarding the Issuing Token Authority:

- The Issuing Token Authority is expected to be configured with policies.
- regarding issuing access tokens providing permission to access resources on the Token Consumer, Access Token Issuing Privileges of the Issuing Token Authority; that is, the range of privileges on the Token Consumer for which the Issuing Token Authority is allowed to issue access token.
- The Issuing Token Authority can be configured with credentials for securing the authorization data received in the access token request (for example, the Token Consumer can encrypt the authorization data for privacy reasons).
- The Issuing Token Authority can be configured with credentials for authentication of the issued access token (that is, the Token Consumer might verify that the Issuing Token Authority genuinely issued this access token) and/or encrypting the access token (for subsequent decryption by the Token Consumer).

Examples of entities that can assume this functional role can include:

- A web server through which the user can approve authorization grants - with the user interface (for approving authorization grants) provided by a web-page or other application on a user device.
- A web server configured with policies for making automated decisions - much like a Policy Decision Point (PDP) in the authorization architecture proposed in oneM2M TR-0016 [i.5].

6.4.2.3 Token Consumer

A Token Consumer performs the following functions within the scope of DAA3:

- Form authorization data to be used by the Issuing Token Authority in deciding the privileges to be granted to the Token Subject in an access token.
- Apply end-to-end security processes to the authorization data if applicable (e.g. encryption, signature generation), with the corresponding security processing to be applied by the Issuing Token Authority.
- Send to the (optionally end-to-end secured) authorization data to the Token Subject, and identify where the access token request is to be submitted (that is, identify the Issuing Token Authority).
- Receive, from a Token Subject, a request to perform actions one or more resources, accompanied by an access token.
- (If the Issuing Token Authority applied end-to-end security to the access token) Apply end-to-end security processes decrypt and or verify that the access token is valid.
- Perform the requested actions, if permitted by the access token.

Assumptions regarding the Token Consumer:

- The Token Consumer is configured with the Access Token Issuing Privileges of the Issuing Token Authority; that is, the range of permissions that the Token Consumer will allow in access tokens issued by the Issuing Token Authority.
- The Token Consumer is configured is configured with the address at which access token requests are to be submitted to the Issuing Token Authority.

- The Token Consumer can be configured with credentials for securing the authorization data (for example, encryption can be applied for privacy reasons).
- The Token Consumer can be configured with credentials for verifying the authenticity of the access token (that is, the Token Consumer might verify that the Issuing Token Authority genuinely issued this access token).
- If the Token Consumer and Issuing Token Authority support use of permission tickets, then the Token Consumer is configured with the address at which permission tickets requests are to be submitted to the Issuing Token Authority.

6.4.3 DAA3 Reference Points

6.4.3.1 DAA3 Token Subject-to-Authority (Dsa) Reference Point

This reference point is between the Token Subject and the Requesting Token Authority.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Token Subject to obtain an access token via the Requesting Token Authority. This can include:
  - The Token Subject providing the Requesting Token Authority with:
    - the information provided by the Token Consumer for requesting a token, that is:
      - authorization data; and
      - identifying where the access token request is to be submitted.
    - A desired time window for the access token.
  - The Requesting Token Authority providing the Token Subject with an access token (received from the Issuing Token Authority) or an error message.

This reference points is presumed be secured hop-by-hop using security associations, and end-to-end-security can optionally be applied.

6.4.3.2 DAA3 Authority-to-Authority (Daa) Reference Point

This reference point is between the Requesting Token Authority and the Issuing Token Authority when these roles are assumed by distinct entities.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Requesting Token Authority to obtain an access token from the Issuing Token Authority. This can include:
  - The Requesting Token Authority providing the Issuing Token Authority with parameters for the access token, including:
    - The authorization data provided by the Token Consumer via the Token Subject.
    - A desired time window for the access token.
  - The Issuing Token Authority requesting (from the Requesting Token Authority) information about the Token Subject that the Issuing Token Authority uses for deciding whether to issue an access token.
  - The Requesting Token Authority providing the Issuing Token Authority with the requested information about the Token Subject.
  - The Issuing Token Authority providing the Requesting Token Authority with an access token or error message.

The reference model assumes that the Issuing Token Authority trusts the Requesting Token Authority have established an arrangement whereby:

- The Issuing Token Authority has an expectation that the Requesting Token Authority to provide correct information about the Token Subject to whom the access token will be provided.
- The Requesting Token Authority can have an expectation that the Issuing Token Authority will maintain confidentiality of information shared about the Token Subject.

The details of such an arrangement are not described in the present document.

This reference point is presumed be secured directly between the Requesting Token Authority and Issuing Token Authority, for example using a TLS security association.

6.4.3.3 DAA3 Token Subject-to-Consumer (Dsc) Reference Point

This reference point is between the Token Subject and the Token Consumer and Issuing Token Authority.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Token Subject to provide an access token with a request to act on a resource(s) to the Token Consumer. This can include:
    - Providing an access token or a unique identifier for the access token (which can be subsequently used by the Token Consumer to retrieve the access token).
    - Enabling the Token Consumer to verify that the Token Subject which sent the request to act on resource(s) is also the Token Subject to whom the access token was issued (for example, using a digital signature or MIC).
- Enabling the Token Consumer to provide an error message to the Token Subject, which can comprise:
    - Identifying that the access token is no longer valid.
    - Identifying an Issuing Token Authority that the Token Consumer allows to issue access tokens. This is used in the case that either no access token was provided, or an access token was provided that can not be used for the associated request. There are a variety of reasons that an access token can not be used:
        - ▪ The provided access token was issued by by an Issuing Token Authority that is not recognized by the Token Consumer.
        - ▪ The provided access token is not currently valid (yet to be valid, or already expired).
        - ▪ The scope of the provided access token does not cover the requested resource(s).

This reference points is presumed be secured hop-by-hop using security associations, and end-to-end-security can optionally be applied.

6.4.3.4 DAA3 Consumer-to-Authority (Dca) Reference Point

This reference point is between the Token Consumer and Issuing Token Authority.

This reference point:

- Defines how the Token Consumer provides authorization data to the Issuing Token Authority to use in composing an access token. This authorization data can be communicated directly form the Token Consumer to the Issuing Token Authority (e.g. if permissions tickets are used), or the authorization data can traverse the Dsc, Dsa and Daa reference points via the Token Subject and Requesting Token Authority.

- Defines how access tokens are formed at the Issuing Token Authority and processed at the Token Consumer. The access tokens can traverse the Dsc, Dsa and Daa reference points via the Token Subject and Requesting Token Authority. Alternatively, the access token data can be communicated directly from the Issuing Token Authority to the Token Consumer, with the Dsc, Dsa and Daa reference points communicating an unique identifier for the access token (rather than communicating the access token data itself). This alternative can suit scenarios where the roles of Issuing Token Authority and Token Consumer are assumed by a single entity.

This reference point can enable some or all of the following features (depending on design choices discussed in clause 7):

- Enabling the Token Consumer to provide the Issuing Token Authority with an authorization data.
- Enabling the Issuing Token Authority to provide the Token Consumer with an access token. This can include:
    - Identify the Token Subject authorized by the access token.
    - Identify the Issuing Token Authority that issued the access token.
    - Verify that the identified Issuing Token Authority issued the access token. This can include providing a MIC or digital signature in the access token.
    - Identify the Token Consumer(s) for which the access token is intended.
    - Identify the permissions represented by the access token.
    - Identify the time window within which the access token is valid.
    - Identify the authorization checks that were carried out by the Issuing Token Authority.
    - Identify the "authData" that was initially provided by the Token Consumer (optional).
- End-to-End security of data exchanged between the Token Consumer and Issuing Token Authority.

6.4.4 Example Procedure of Access Token Issuance and Use in DAA3

This clause provides an example message flow for DAA3 Access Token issuance and use in which the Token Consumer communicates the authorization data to the Issuing Token Authority via the Originator and Requesting Token Authority. Other alternatives are possible, such as discussed in clause 6.4.3.4 "DAA3 Consumer-to-Authority (Dca) Reference Point". Most message flows will include steps similar to steps 4 through to 15. Optimizations to this message flow can be considered in Clause 8.

The descriptions of functional roles in Clause 6.4.2 "DAA3 Functional Roles" and reference points in Clause 6.4.3 "DAA3 Reference Points" provide assumptions which are pre-requisites for the generic procedure of token issuance in the DAA3 reference model.

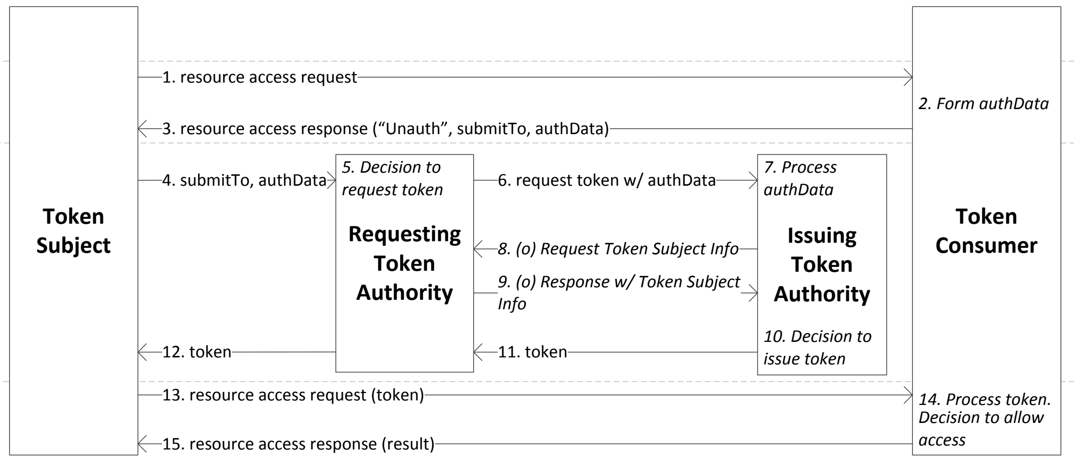

Token Subject
Token Consumer
Requesting Token Authority
Issuing Token Authority
1. resource access request
2. Form authData
3. resource access response (“Unauth”, submitTo, authData)
4. submitTo, authData
5. Decision to request token
6. request token w/ authData
7. Process authData
8. (o) Request Token Subject Info
9. (o) Response w/ Token Subject Info
10. Decision to issue token
11. token
12. token
13. resource access request (token)
14. Process token. Decision to allow access
15. resource access response (result)

Figure 6.4.4-1: General Procedure for Token Issuance and Use in the DAA3 reference model.

The generic procedure of token issuance and use is shown in figure 6.4.4-1 and described as follows:

NOTE: This message flow borrows details from the message flow in figure 6.3.2-1.

A. Setup (Not Shown - see description in clause 6.4.1 "Overall Description").

B. Initialization (Optional - if the Token Subject already knows submitTo and authData, then these steps can be skipped)

1) The Token Subject sends the Token Consumer a resource access request, possibly including one or more access tokens. For the purpose of this description, it is assumed that, even with these access tokens, the Token Subject does not have sufficient privileges for the Token Consumer to process the resource access request.

2) The Token Consumer determines that the Token Subject currently does not have sufficient privileges for the Token Consumer to process the resource access request. The Token Consumer forms an authorization data describing, for the Issuing Token Authority, a set of privileges to be provided to the Token Subject. The Token Consumer can apply end-to-end security processes to the authorization data if applicable (e.g. encryption, signature generation).

3) The Token Consumer sends a resource access response to the Token Subject containing:

   - An indication that the request does not have sufficient authorization.

   - submitTo: identifying where the access token request is submitted. The Token Consumer is presumed to have been configured with this address.

   - The authorization data (authData), optionally secured end-to-end (with the corresponding security processing to be applied by the Issuing Token Authority).

C. Obtaining an Access Token

1) The Token Subject forwards submitTo and (optionally end-to-end secured) authorization data to the Requesting Token Authority.

2) The Requesting Token Authority applies its policies to decide if the Requesting Token Authority submits a token request to the Issuing Token Authority on behalf of the Token Subject.

3) The Requesting Token Authority sends the Issuing Token Authority a token request containing the (optionally end-to-end secured) authorization data.

4) If the authorization data is secured end-to-end (see step 2), then the Issuing Token Authority applies appropriate processing to decrypt and/or verify integrity of the authorization data.

NOTE 1: The Issuing Token Authority can be configured to reject tokens that are not secured end-to-end by the Token Consumer.

The Issuing Token Authority examines its policies to determine if additional information (about the Token Subject and/or Requesting Party) is used in its decision to issue a token covering this authorization data.

5) (Optional) The Issuing Token Authority sends the Requesting Token Authority a request for additional information about the Token Subject

6) (Conditional on Step 8) The Requesting Token Authority sends the Issuing Token Authority additional information about the Token Subject.

7) The Issuing Token Authority does the following operations:

NOTE 2: The remaining steps assume that the Issuing Token Authority decided to allow issuing the access token.

   1) Check if the Token Subject has the privileges of accessing the target resource according to the ACP.

   2) Check if the requested privileges are within the Access Token Issuing Privileges of the Issuing Token Authority. This operation can be performed by a Token Authorization Function on behalf of the Issuing Token Authority.

   3) Check if the Token Subject can obtain the privileges described in the Access Token Request according to the Access Token Authorization Policies. This operation can be performed by a Token Authorization Function on behalf of the Issuing Token Authority.

    4) Generate access token plaintext giving the Token Subject the privileges described in authorization data received in Step 6.

    5) Sign and/or encrypt the access token plaintext.

8) The Issuing Token Authority sends the Requesting Token Authority the Access Token.

9) It is also possible for the Issuing Token Authority to store the Access Token and inform the Token Subject where to retrieve the Access Token.

10) The Requesting Token Authority sends the Token Subject the access token it received form the Issuing Token Authority.

D. Accessing a resource using an Access Token

1) The Token Subject sends the Token Consumer a resource access request, this time including the access token it received via the Requesting Token Authority.

2) The Token Consumer extracts the Access Tokens from the resource access request, and does the following operations:

    1) Decrypt and/or verify the Access Token.

    2) Check if the privileges in the Access Token are within the Access Token Issuing Privileges of the Issuing Token Authority.

    3) Check if this Access Token has been revoked against an Access Token Revocation List. Also checks to see if the current time is within the validity of the token.

    4) Evaluate the Token Subject's resource access request based on the privileges in the Access Token.

    5) Perform the requested resource access.

NOTE 3: The Token Consumer can be configured to reject tokens that are not secured end-to-end by the Issuing Token Authority.

3) The Token Consumer returns the execution result back to the Token Subject.

6.5 Dynamic Authorization Architecture Proposal 4 (DAA4) Reference Model

6.5.1 Overall Description

Table 6.5.1-1 lists the functional roles in the DAA4 reference model. Clause 6.5 describes the functions associated with these functional roles.

Table 6.5.1-1: List of DAA4 functional roles

DAA4 Functional Role	Description	Details in clause
Subject	The entity being provided with authorization to access resources on the Hosting Entity. The Subject would be authorized dynamically by an Authorization Entity in order to access resources.	6.5.2.1
Hosting Entity	This entity interacts with the Subject in order to obtain information about the Subject that can be used for authorization checks. This entity interacts with an Authorization entity to provide information about the Subject and request dynamic authorization checks on the Subject.	6.5.2.2
Authorization Entity	This entity performs dynamic authorization checks of the Subject based on the information that was forwarded by a Hosting Entity. This entity sends a response indicating authorization results and additional information that can be used to update the access control policies at the Hosting Entity. This entity can belong to a different operator than the Hosting Entity and can be multiple hops away from the Hosting Entity.	6.5.2.3

The DAA4 reference model assumes the following sequence of events take place.

0) **Setup:**

   - The Authorization Entity is provided with policies governing processing of authorization requests and policies governing the issuing of authorization responses.
   - The Authorization Entity is pre-configured with Subject's identity and information that can be used for performing the authorization checks.
   - The Authorization Entity is provided with policies governing the authorization checks that will be carried out on the Subject.
   - The Hosting Entity is provided with the information about the Authorization Entity and pre-configured with credentials that can be used to perform mutual authentication between the Hosting Entity and the Authorization Entity. Similarly the Authorization Entity is pre-configured with the information associated with the Hosting Entity and associated credentials that can be used to perform mutual authentication between the Hosting and the Authorization entities.
   - The Hosting Entity is provided with dynamic authorization policies, that governs decisions on requesting dynamic authorization request, processing dynamic authorization results from the Authorization Entity and updating of access control policies based on the authorization results.
   - The Subject is optionally provided with security parameters that can be used for authorization checks.

1) **Accessing a resource:** Subject sends a resource access request to the Hosting Entity. The Hosting Entity checks the access control policies and if there is no access policies associated with the subject then the Hosting Entity initiates a dynamic authorization check and as per the set policies forwards an authorization request to the Authorization Entity.

2) **Authorization Checks:** Based on the authorization request forwarded by the Hosting Entity, the Authorization Entity performs one or more authorization checks based on the Subject's information that was provided within the authorization request. The results of the authorization checks are then forwarded to the Hosting Entity by the Authorization Entity containing information about the resource, type of operation(s) and the validity of the authorization. The Hosting Entity updates the ACPs accordingly based on the results provided to it by the Authorization Entity.

6.5.2 DAA4 Functional Roles

6.5.2.1 Subject

A Subject performs the following functions within the scope of DAA4:

- Perform a resource access with a Hosting Entity.
- (Optional) The Subject can be involved in authorization checks with an Authorization Entity based on the type of authorization checks that will be carried out.

6.5.2.2 Hosting Entity

Within the scope of DAA4, a Hosting Entity is able to perform the following functions:

- Initiate a dynamic authorization check if an access control policy associated with the Subject does not exist. The initiation of the dynamic authorization check is determined by pre-provisioned policies.
- The Hosting Entity communicates with an Authorization Entity in order to request for dynamic authorization of the Subject's request for a resource access. The request created contains information about the Subject, including the Subject's identity, and optionally, the resource, resource type and the type of operations.
- The Hosting Entity processes the response containing the results of the authorization check from the Authorization Entity and updates the relevant access control policies accordingly based on the results. The Hosting Entity is able to verify the authenticity of the authorization response message.

6.5.2.3 Authorization Entity

Within the scope of DAA4, an Authorization Entity is able to perform the following functions:

- It is able to process the authorization request sent by the Hosting Entity.
- Based on the dynamic authorization policies as well as the Subject's information that was provided as well as the information provided with regards to the resource and associated operations, the Authorization Entity initiates one or more authorization checks of the Subject.
- Ability to create an authorization response containing the results of the authorization checks and send it to the Hosting Entity. The Authorization Entity can include an authentication tag (e.g. MIC) or a digital signature vouching for the authenticity of the authorization.

6.5.3 DAA4 Reference Points

The DAA4 uses the oneM2M standardized reference points:

- Mca: Used for communications between the Subject and the Hosting Entity. This reference point is optionally protected by means of a (D)TLS connection.
- Mcc: Used for communications between the Hosting Entity and the Authorization Entity. This reference point is protected by means of hop-by-hop (D)TLS connection. In addition, end-to-end security protections can be used if the Hosting Entity and the Authorization Entity belongs to different service provider domains or if the Entities are multiple hops away.

6.5.4 Example Procedure of Dynamic Authorization using oneM2M reference points and primitives

This clause provides example message flow for DAA4. The oneM2M reference points and primitives are used with having very little impact on the overall oneM2M messaging and associated resource for dynamic authorization.

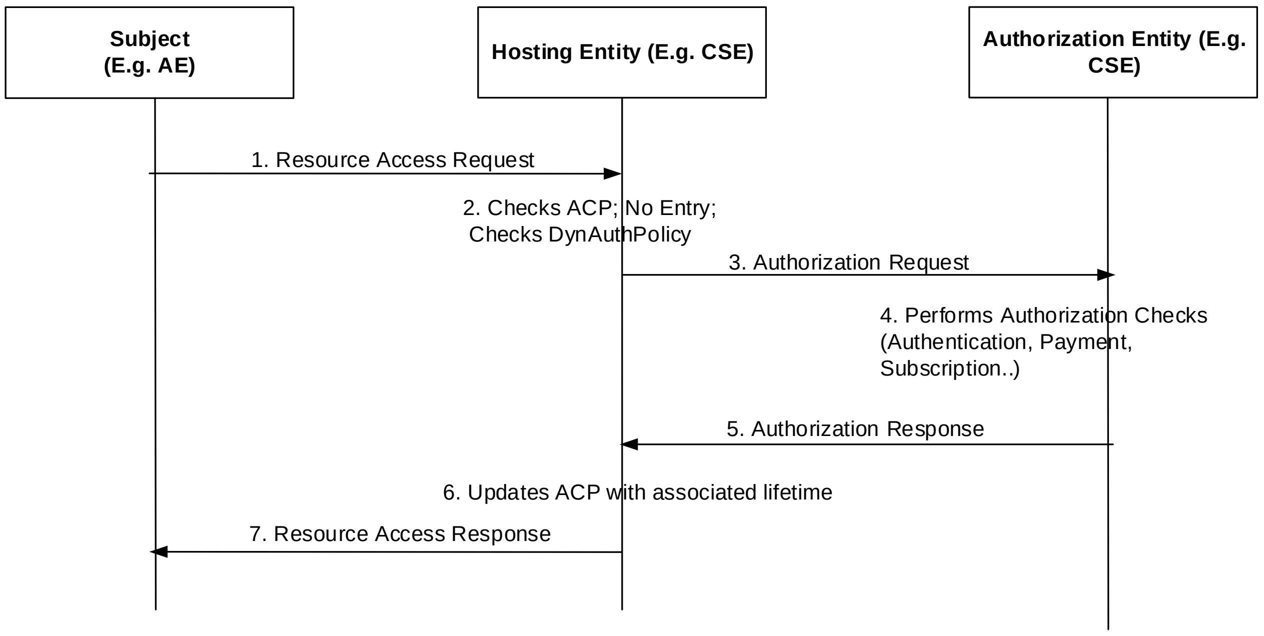

Subject
(E.g. AE)
Hosting Entity (E.g. CSE)
Authorization Entity (E.g. CSE)
1. Resource Access Request
2. Checks ACP; No Entry; Checks DynAuthPolicy
3. Authorization Request
4. Performs Authorization Checks (Authentication, Payment, Subscription..)
5. Authorization Response
6. Updates ACP with associated lifetime
7. Resource Access Response

Figure 6.5.4-1: General Proposal for Dynamic Authorization 4 (DAA4) Reference Model

The generic procedure for dynamic authorization based on oneM2M reference points and primitives shown in figure 6.5.4-1. The detailed procedure is as follows:

1) A Subject issues a Resource Access Request to a Hosting Entity over the Mca reference point.

2) The Hosting Entity checks the ACPs associated with the Subject and if there does not exist an ACP associated with the request, then the Hosting Entity checks the Dynamic Authorization Policy (DynAuthPolicy). If there is a dynamic authorization policy that can be identified based on the request, then the Hosting Entity initiates a dynamic authorization process.

3) The Hosting Entity sends an Authorization Request to an Authorization Entity, requesting for authorization of the Subject and optionally the resource information, the resource-type, type of operations etc. are included within the Authorization Request.

4) The Authorization Entity based on the Authorization Request can perform one or more authorization checks such as Subject authentication, payment verification, subscription information etc. and then generates an authorization result.

5) The Authorization Entity sends the authorization results to the Hosting Entity using an authorization response message. The authorization results can contain the duration of the validity of the authorization. The message is provided with an authentication tag or digital signature particularly if the Authorization Entity is either multiple hops away from the Hosting Entity or if the Authorization Entity belongs to a different service provider domain.

6) The Hosting Entity verifies the authenticity of the message and then processes the authorization response. The Hosting Entity updates the local ACP and includes an optional time validity associated with the access control rule.

7) If the authorization results provided by the Authorization Entity matched the resource access request, then the Hosting Entity sends a Resource Access Response message to the Subject that indicates a successful authorization.

6.6 Dynamic Authorization Architecture Proposal 5 (DAA5) Reference Model

6.6.1 Token Based Access Control Architecture

Figure 6.6.1-1 provides a high level overview of the token based access control architecture in the oneM2M system. This architecture comprises two new defined entities that are described as follows:

- Token Authority: It is responsible for issuing access tokens to an Originator according to some privilege assignment policies. These policies are however out of scope of the present document. An access token specifies what privileges have been assigned to an Originator. The Originator can be an AE or a CSE.

- External Authorization Function: It is responsible for evaluating a privilege authorization request sent by a Token Authority and then making a privilege authorization decision. How an External Authorization Function being implemented is however out of scope of the present document.

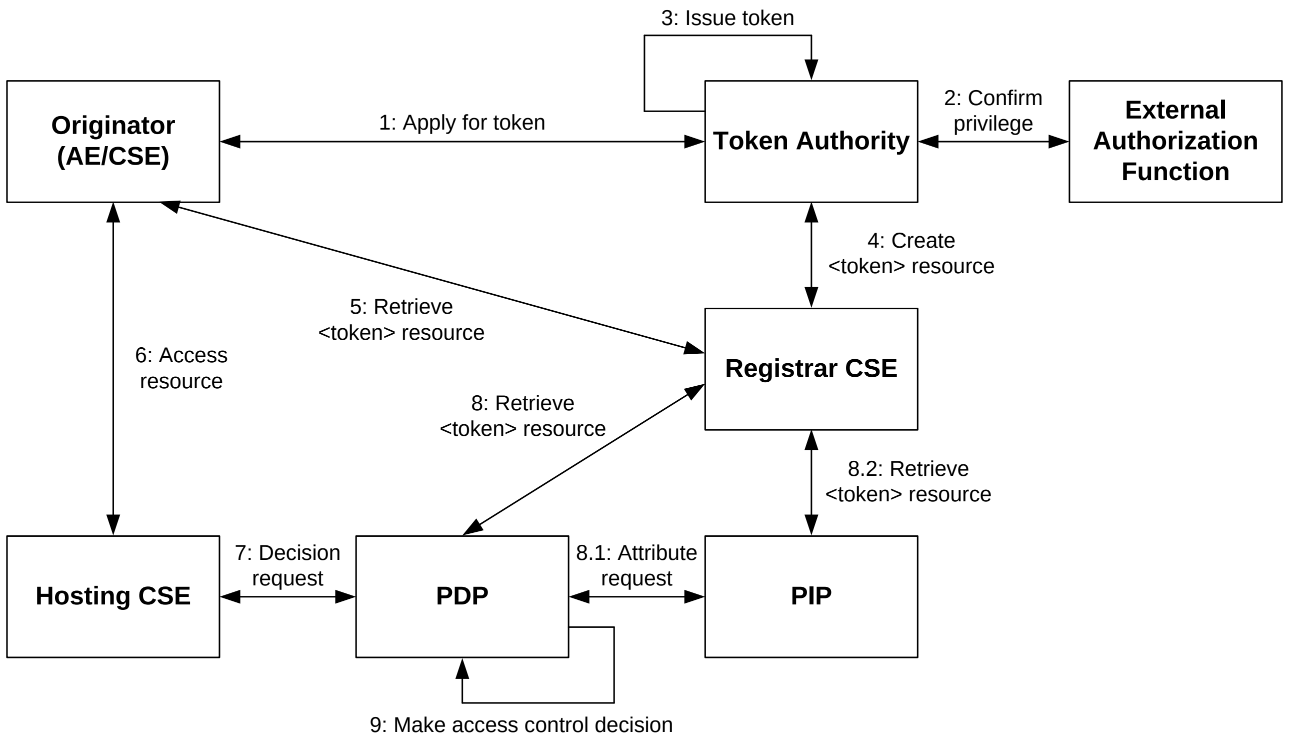


Figure 6.6.1-1: Token based access control architecture

6.6.2 Generic Procedure of Token Based Access Control

The generic token based access control procedure is shown in figure 6.6.1-1 and described as follows:

Step 001:	An Originator applies for a token from a Token Authority.
Step 002:	The Token Authority checks if the applied privilege can be assigned to the Originator. In case the Token Authority is not sure whether to agree to the request locally, it can contact an External Authorization Function for checking this request.
Step 003:	After confirming the Originator's request, the Token Authority issues a token to the Originator.
Step 004:	The Token Authority create a <token> resource that is used to store the token information under the Originator's registration resource in the Originator's Registrar CSE.
Step 005:	The Originator retrieves assigned tokens from its registration resource.
Step 006:	The Originator sends a resource access request in which tokens are included to the Hosting CSE.
Step 007:	The Hosting CSE sends access decision request to a PDP.
Step 008:	In some cases, the PDP retrieves the Originator's token from the Originator's registration resource.
Step 009:	The PDP verifies the Originator's token and makes an access control decision according to access control policies and the privileges described in the token.

7 Description and Analysis of Available Options

7.1 Proposal 1: A Solution of Access Token Issuance and Use

7.1.1 Resource Definitions for Token Issuance

7.1.1.1 Resource Type *accessTokenAuthority*

The <*accessTokenAuthority*> resource represents the method for providing the services related to access token issuance. The <*accessTokenAuthority*> resource is located directly under <*CSEBase*>.

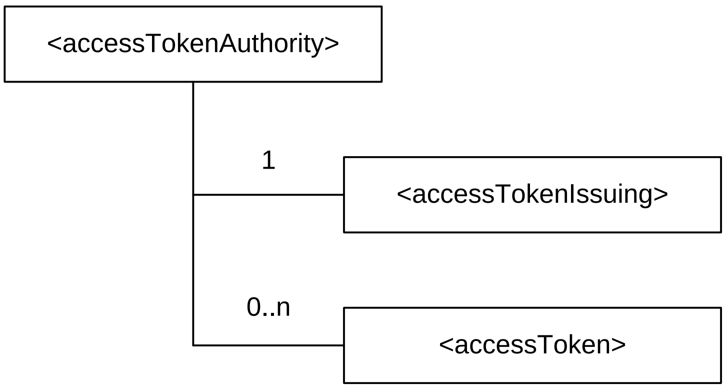


Figure 7.1.1.1-1: Structure of <*accessTokenAuthority*> resource

The <*accessTokenAuthority*> resource contains the child resources specified in table 7.1.1.1-1.

Table 7.1.1.1-1: Child resources of <*accessTokenAuthority*> resource

Child Resources of <*authorization*>	Child Resource Type	Multiplicity	Description
[variable]	<*accessTokenIssuing*>	1	See clause 7.1.1.2
[variable]	<*accessToken*>	0..n	See clause 7.1.1.3

The <*accessTokenAuthority*> resource contains the attributes specified in table 7.1.1.1-2.

Table 7.1.1.1-2: Attributes of <*accessTokenAuthority*> resource

Attributes of <*statsConfig*>	**Multiplicity**	**RW/ RO/ WO**	**Description**
resourceType	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
resourceID	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
resourceName	1	WO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
parentID	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
authorizationPolicyIDs	1 (L)	RW	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
creationTime	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
expirationTime	1	RW	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
lastModifiedTime	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
labels	0..1 (L)	RW	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described

7.1.1.2 Resource Type *accessTokenIssuing*

The <*accessTokenIssuing*> resource is a virtual resource because it does not have a representation. It is the child resource of the <*accessTokenAuthority*> resource. When a CREATE Request addresses the <*accessTokenAuthority*> resource, an access token issuance process is triggered. The access token request is included in the Content parameter of the CREATE Request, and the access token response is included in the Content parameter of the CREATE Response.

The <*accessTokenIssuing*> resource inherits access control policies that apply to the parent <*accessTokenAuthority*> resource.

7.1.1.3 Resource Type *accessToken*

The <*accessToken*> resource represents M2M Access Token information that is created by an access token insurance process. The <*accessToken*> resource is the child resource of the <*accessTokenAuthority*> resource.

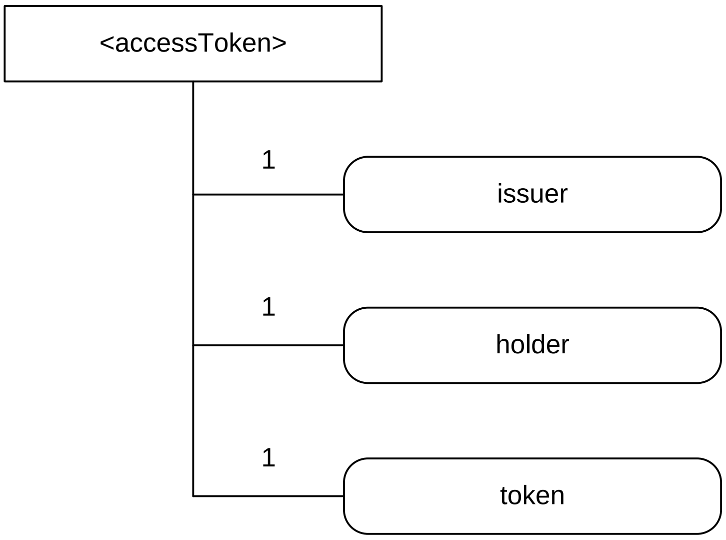


Figure 7.1.1.3-1: Structure of <*accessTokenAuthority*> resource

The <*accessToken*> resource contains the attributes specified in table 7.1.1.3-1.

Table 7.1.1.3-1: Attributes of <*accessToken*> resource

Attributes of <*statsConfig*>	**Multiplicity**	**RW/ RO/ WO**	**Description**
resourceType	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
resourceID	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
resourceName	1	WO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
parentID	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
creationTime	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
expirationTime	1	RW	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
lastModifiedTime	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
labels	0..1 (L)	RW	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described
issuer	1	RO	This attribute contains the identifier of the token issuer
holder	1	RO	This attribute contains the identifier of the token holder
token	1	RO	This attribute contains the access token itself

7.1.2 Resource Procedures for Token Issuance

7.1.2.1 <*accessTokenAuthority*> Resource Procedures

7.1.2.1.1 Introduction

This clause describes the management procedures for the <*accessTokenAuthority*> resource and its child resources.

7.1.2.1.2 Create <*accessTokenAuthority*>

This procedure is used to create a <*accessTokenAuthority*> resource.

Table 7.1.2.1.2-1: <*accessTokenAuthority*> CREATE

<*accessTokenAuthority*> CREATE	
Associated Reference Point	Mcc and Mcc'
Information in Request message	All parameters defined in table 8.2.2-2 of oneM2M TS-0001 [i.2] apply with the specific details for: ***To:*** the address of the <*CSEBase*> where the <*accessTokenAuthority*> resource is intended to be Created. ***Content:*** attributes of the <*accessTokenAuthority*> resource as defined in clause 7.1.1.1-2
Processing at Originator before sending Request	According to clause 10.1.1.1 of oneM2M TS-0001 [i.2]
Processing at Receiver	According to clause 10.1.1.1 of oneM2M TS-0001 [i.2]with the following additions: • Upon successful validation of the provided attributes, the Hosting CSE creates the <*accessTokenAuthority*> resource including its virtual child resource specified in table 7.1.1.1-1 • If there is a access token issuance process to be bound to the <*accessTokenIssuing*> virtual resource, then bind it to the <*accessTokenIssuing*> virtual resource, otherwise leave the binding void. The access token issuance process and the binding method are out of scope
Information in Response message	According to clause 10.1.1.1 of oneM2M TS-0001 [i.2]
Processing at Originator after receiving Response	According to clause 10.1.1.1 of oneM2M TS-0001 [i.2]
Exceptions	According to clause 10.1.1.1 of oneM2M TS-0001 [i.2]

7.1.2.1.3 Retrieve <*accessTokenAuthority*>

This procedure is used to retrieve the attributes of a <*accessTokenAuthority*> resource.

Table 7.1.2.1.3-1: <*accessTokenAuthority*> RETRIEVE

<*accessTokenAuthority*> RETRIEVE	
Associated Reference Points	Mcc and Mcc'
Information in Request message	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Processing at Originator before sending Request	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Processing at Receiver	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Information in Response message	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Processing at Originator after receiving Response	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Exceptions	According to clause 10.1.2 of oneM2M TS-0001 [i.2]

7.1.2.1.4 Update <*accessTokenAuthority*>

This procedure is used to update the attributes of a <*accessTokenAuthority*> resource.

Table 7.1.2.1.4-1: <*accessTokenAuthority*> UPDATE

<*accessTokenAuthority*> UPDATE	
Associated Reference Points	Mcc and Mcc'
Information in Request message	According to clause 10.1.3 of oneM2M TS-0001 [i.2]
Processing at Originator before sending Request	According to clause 10.1.3 of oneM2M TS-0001 [i.2]
Processing at Receiver	According to clause 10.1.3 of oneM2M TS-0001 [i.2]
Information in Response message	According to clause 10.1.3 of oneM2M TS-0001 [i.2]
Processing at Originator after receiving Response	According to clause 10.1.3 of oneM2M TS-0001 [i.2]
Exceptions	According to clause 10.1.3 of oneM2M TS-0001 [i.2]

7.1.2.1.5 Delete <*accessTokenAuthority*>

This procedure is used to delete a <*accessTokenAuthority*> resource.

Table 7.1.2.1.5-1: <*accessTokenAuthority*> DELETE

<*accessTokenAuthority*> DELETE	
Associated Reference Points	Mcc and Mcc'
Information in Request message	According to clause 10.1.4 of oneM2M TS-0001 [i.2]
Processing at Originator before sending Request	According to clause 10.1.4 of oneM2M TS-0001 [i.2]
Processing at Receiver	According to clause 10.1.4 of oneM2M TS-0001 [i.2]
Information in Response message	According to clause 10.1.4 of oneM2M TS-0001 [i.2]
Processing at Originator after receiving Response	According to clause 10.1.4 of oneM2M TS-0001 [i.2]
Exceptions	According to clause 10.1.4 of oneM2M TS-0001 [i.2]

7.1.2.2 <*accessTokenIssuing*> Resource Procedures

7.1.2.2.1 Introduction

This clause describes the management procedures for the <*accessTokenIssuing*> resource. This virtual resource is used to trigger an access token issuance process. Only the Create operation is allowed on this virtual resource.

7.1.2.2.2 Create <*accessTokenIssuing*>

This procedure is used to trigger an access token issuance process that is bound to a <*accessTokenIssuing*> virtual resource.

Originator: The Originator requests an access token by using CREATE operation on a <*accessTokenIssuing*> virtual resource for a <*accessToken*>. The Originator is an AE or a CSE. The Originator provide the information about what privileges the Originator wants to apply.

Receiver: The Receiver checks if the Originator has CREATE permission on the <*accessTokenIssuing*> virtual resource. Upon successful validation, the Receiver check what privileges can be included in the access token according to the authorization policies, and then generate an access token for the Originator. If there is no process bound to the <*accessTokenIssuing*> virtual resource, then the Receiver responds with an error.

Table 7.1.2.2.2-1: <*accessTokenIssuing*> CREATE

<*accessTokenIssuing*> CREATE	
Associated Reference Points	Mcc and Mcc'
Information in Request message	According to clause 10.1.2 of oneM2M TS-0001 [i.2] with the following additions: ***From:*** Identifier of the AE or the CSE that initiates the Request ***To:*** The address of the <*accessTokenIssuing*> virtual resource ***Content:*** The representation of the access token request
Processing at Originator before sending Request	The Originator requests creation of a <*accessToken*> resource by using the CREATE operation. The request addresses the <*accessTokenIssuing*> virtual resource under a <*accessTokenAuthority*> of a Hosting CSE. The Originator is an AE or a CSE. The originator provides the information about what privileges the Originator want to apply. The access token request is included in the Content parameter of the Request message. The Originator can be an AE or a CSE.
Processing at Receiver	The Receiver performs the following operations: • Check if the Originator has CREATE permission under the target <*accessTokenIssuing*> virtual resource • Check the validity of the provided parameters • Check if the <*accessTokenIssuing*> virtual resource is bound to an access token process • Check if the requested privileges are within the Access Token Issuing Privileges • Upon successful validation, check what privileges can be agreed and included access token according to access token authorization policies • Sign and/or encrypt the access token plaintext • Create a <*accessToken*> resource for the generated access control token under the <*accessTokenAuthority*> resource • Create an access token response in which either the generated access token or the resource address of the created <*accessToken*> resource is included and send it to the Originator. The access token response is included in the Content parameter of the Response message
Information in Response message	According to clause 10.1.2 of oneM2M TS-0001 [i.2] with the following additions: ***Content:*** The representation of the access token response
Processing at Originator after receiving Response	According to clause 10.1.1.1 of oneM2M TS-0001 [i.2]
Exceptions	According to clause 10.1.2 of oneM2M TS-0001 [i.2] with the following: • There is no token issuance process bound to the <*accessTokenIssuing*> virtual resource • The provided content of the access token request is not in line with the specified structure

7.1.2.3 <*accessToken*> Resource Procedures

7.1.2.3.1 Introduction

This clause describes the management procedures for the <*accessToken*> resource. This resource is used to store the access token generated by the access token process. Only the Retrieve operation is allowed on this resource.

7.1.2.3.2 Retrieve <*accessToken*>

This procedure is used to retrieve attributes information of a <*accessToken*> resource. The generic retrieve procedure is described in clause 10.1.2 of oneM2M TS-0001 [i.2].

Table 7.1.4.3.2-1: <*accessToken*> RETRIEVE

<*accessToken*>RETRIEVE	
Associated Reference Point	Mca, Mcc and Mcc'
Information in Request message	All parameters defined in table 8.2.2-2 of oneM2M TS-0001 [i.2]apply with the specific details for: • ***Content:*** void
Processing at Originator before sending Request	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Processing at Receiver	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Information in Response message	All parameters defined in table 8.2.3-1 of oneM2M TS-0001 [i.2] apply with the specific details for: • ***Content*:** attributes of the <*subscription*> resource as defined in clause 8.2.3.3
Processing at Originator after receiving Response	According to clause 10.1.2 of oneM2M TS-0001 [i.2]
Exceptions	According to clause 10.1.2 of oneM2M TS-0001 [i.2]

7.2 Proposal 4: oneM2M-based SLDA

7.2.1 Introduction

The following clauses propose an architecture, procedures, and resource definitions for enabling the oneM2M system to support Service Layer Dynamic Authorization (SLDA) functionality.

A summary of the proposed oneM2M enhancements include:

1) Enhancements to the oneM2M architecture to support SLDA functionality and to show which types of oneM2M entities SLDA functionality can be included within (e.g. MEF, MAF and CSEs).

2) The definition of several oneM2M procedures providing message sequence level descriptions of how different types of dynamic authorization functionality can be supported within a oneM2M system.

3) The definition of a oneM2M <*dynAuthPolicy*> resource and attributes to support configuration of service layer dynamic authorization policies.

4) The definition of a oneM2M <*consult*> resource and corresponding request and response message formats to support the ability to perform consultation-based dynamic authorization.

5) Some proposed enhancements to the existing oneM2M defined <*accessControlPolicy*> resource to include an identifier for an individual privilege within the privilege list that enables partial addressing of individual privileges within privileges list, an expiration time for each individual privilege, and the definition of new types of authorization context that support use cases involving payment-based, reputation-based, and security assessment-based dynamic authorization.

7.2.2 Architecture

SLDA functionality can optionally be hosted on various oneM2M defined entities including a CSE, a MEF or a MAF. SLDA functionality can be centralized and hosted in its entirety on one of these oneM2M defined entities, or alternatively, the SLDA functionality can be partitioned such that its sub-functions (SLDA-PA, SLDA-PD, SLDA-PE and SLDA-PI) can be hosted in a distributed manner across multiple oneM2M entities.

EXAMPLE: The SLDA-PI sub-function can be hosted on a MEF, the SLDA-PD and SLDA-PA sub-functions on a MAF and the SLDA-PE sub-function on a CSE. When hosted in a distributed manner, the SLDA sub-functions can coordinate with one another to perform dynamic authorization functions.

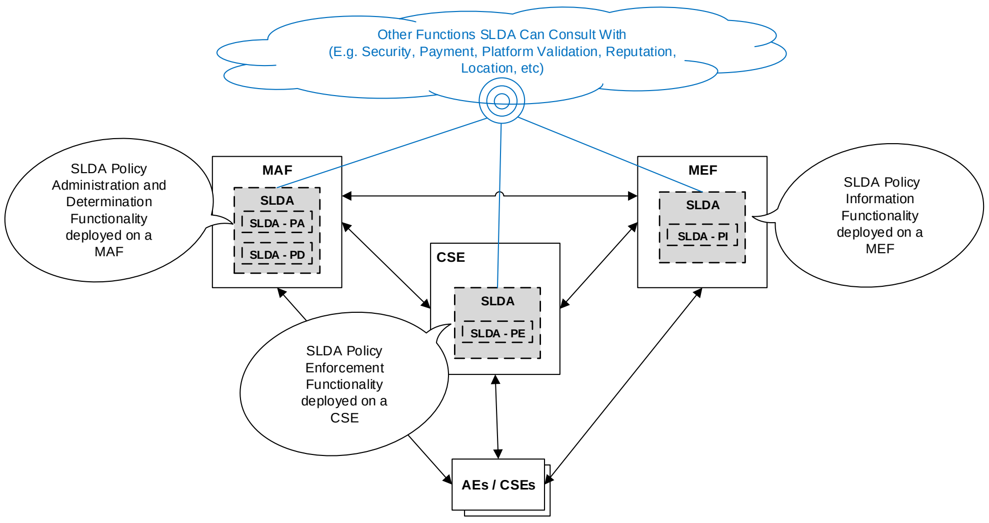

Other Functions SLDA Can Consult With
(E.g. Security, Payment, Platform Validation, Reputation, Location, etc)
SLDA Policy Administration and Determination Functionality deployed on a MAF
MAF
SLDA
SLDA - PA
SLDA - PD
MEF
SLDA
SLDA - PI
SLDA Policy Information Functionality deployed on a MEF
CSE
SLDA
SLDA - PE
SLDA Policy Enforcement Functionality deployed on a CSE
AEs / CSEs

Figure 7.2.2-1: SLDA deployment distributed across oneM2M MEF, MAF and CSE Entities

7.2.3 General Procedure for Dynamic Authorization Policy Provisioning and Configuration, Policy Evaluation and Enforcement

7.2.3.1 Configuration of Dynamic Authorization Policies

Dynamic authorization policies can be configured via creating/updating/deleting the *dynAuthRules* attribute of a targeted <*dynAuthPolicy*> resource. Once configured, an SLDA function can make use of these policies to perform dynamic authorization request handling.

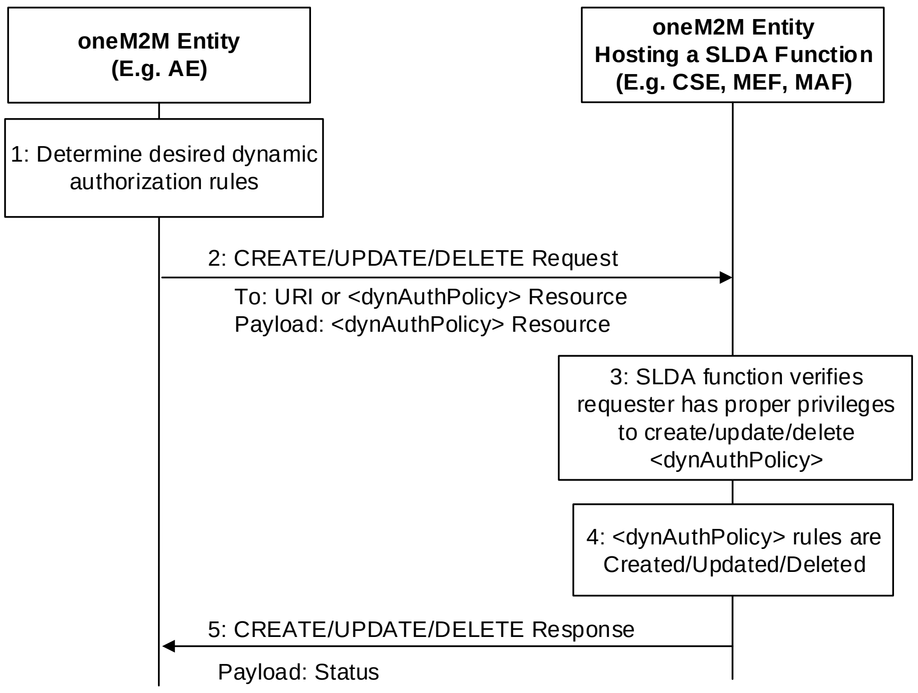

oneM2M Entity (E.g. AE)
oneM2M Entity Hosting a SLDA Function (E.g. CSE, MEF, MAF)
1: Determine desired dynamic authorization rules
2: CREATE/UPDATE/DELETE Request
To: URI or <dynAuthPolicy> Resource
Payload: <dynAuthPolicy> Resource
3: SLDA function verifies requester has proper privileges to create/update/delete <dynAuthPolicy>
4: <dynAuthPolicy> rules are Created/Updated/Deleted
5: CREATE/UPDATE/DELETE Response
Payload: Status

Figure 7.2.3.1-1

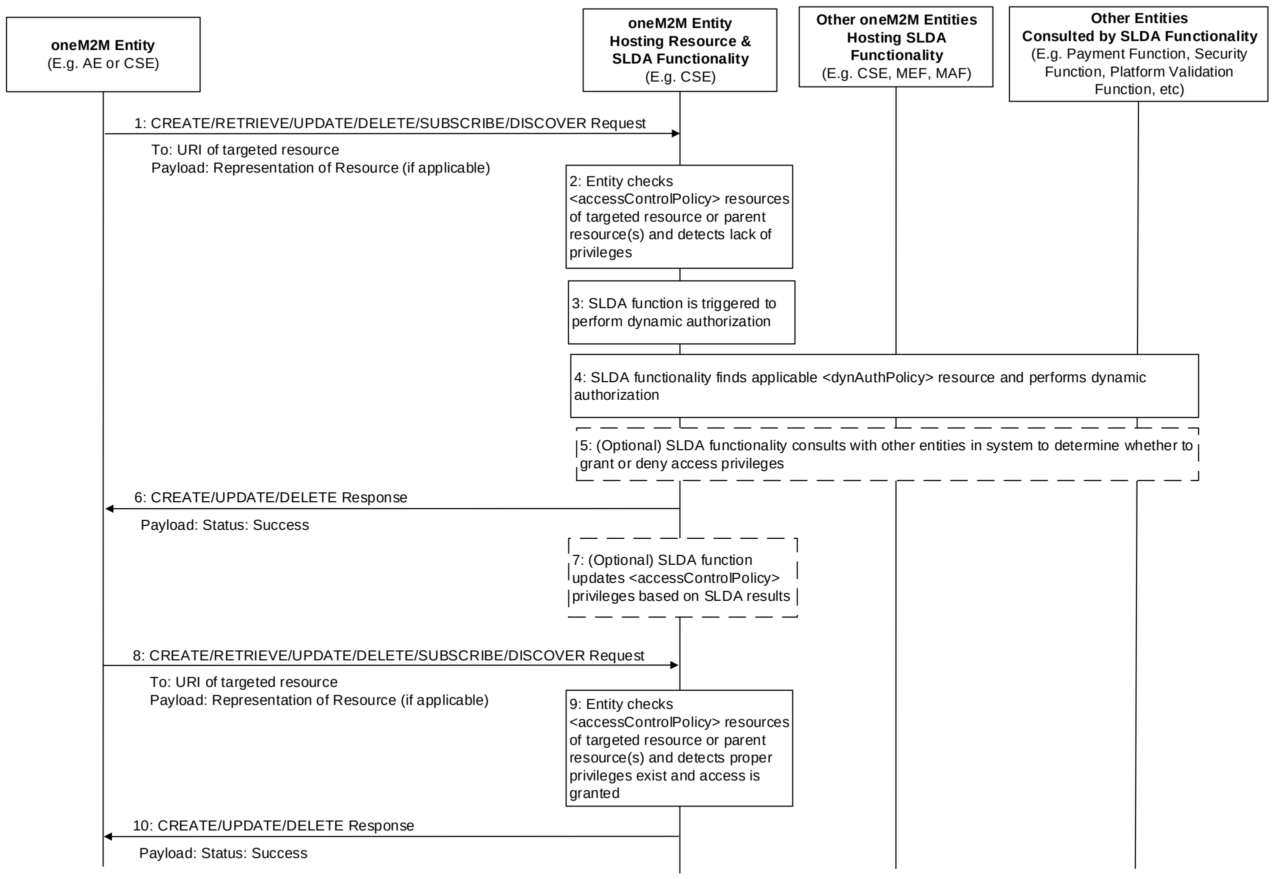
- **Step 1:** A oneM2M entity (e.g. AE) determines the type of dynamic authorization rule it would like to configure and the corresponding values for this rule (such as consultation-based rules, payment-based rules, security-assessment-based rules and reputation-based rules).
- **Step 2:** A oneM2M entity (e.g. AE) sends request targeting a second oneM2M entity (e.g. CSE, MEF or MAF) that hosts a SLDA function. The request contains a <*dynAuthPolicy*> resource representation, where the *dynAuthRules* attribute of the <*dynAuthPolicy*> resource can be configured with dynamic authorization policy rules from Step 1.
- **Step 3:** The SLDA function hosted on the oneM2M entity receives the request and checks whether the requester has the proper privileges to perform the specified operation on the targeted <*dynAuthPolicy*> resource. This check is performed by confirming whether the requester has configured *privileges* within the corresponding <*accessControlPolicy*> resource associated with the targeted <*dynAuthPolicy*>. If yes, the SLDA function continues with processing the request, otherwise the SLDA function returns an error to the requester indicating lack of privileges to perform this operation to the requester.
- **Step 4:** The SLDA function processes the request and based on the operation either creates, updates or deletes the specified rules to/from the *dynAuthRules* attribute of the targeted <*dynAuthPolicy*> resource.
- **Step 5:** The receiving oneM2M entity returns a response to the requester indicating whether or not the request to create, update, or delete the specified rules in the *dynAuthRules* attribute of the targeted <*dynAuthPolicy*> resource was successful or not.

7.2.2.2 Service Layer Dynamic Authorization

7.2.2.2.1 Introduction

Service Layer Dynamic Authorization can be triggered and performed autonomously by a oneM2M entity that hosts a SLDA function. Via this procedure an SLDA function can support autonomously granting or denying privileges on-the-fly during the processing of requests made to access resources which a requester does not have existing privileges to access. By supporting this procedure, burden is removed from requesters since they do not explicitly request access privileges for resources they do not have privileges established for. Via this procedure, service layer dynamic authorization can be performed without requesters being aware that it is being performed. This reduces the overhead on requesters especially for use cases involving resource constrained devices (e.g. sensor applications hosted on IoT devices).

This procedure is also beneficial for the service layer itself since it offers a mechanism for access control policies to be dynamically created or updated on-the-fly leveraging SLDA results rather than having to have an external management entity in the network statically pre-configure access control policies with a set of privileges in advance. Pre-configuring access control policies in advance can be burdensome, inflexible and not very scalable.



oneM2M Entity
(E.g. AE or CSE)
oneM2M Entity
Hosting Resource &
SLDA Functionality
(E.g. CSE)
Other oneM2M Entities
Hosting SLDA
Functionality
(E.g. CSE, MEF, MAF)
Other Entities
Consulted by SLDA Functionality
(E.g. Payment Function, Security
Function, Platform Validation
Function, etc)
1: CREATE/RETRIEVE/UPDATE/DELETE/SUBSCRIBE/DISCOVER Request
To: URI of targeted resource
Payload: Representation of Resource (if applicable)
2: Entity checks <accessControlPolicy> resources of targeted resource or parent resource(s) and detects lack of privileges
3: SLDA function is triggered to perform dynamic authorization
4: SLDA functionality finds applicable <dynAuthPolicy> resource and performs dynamic authorization
5: (Optional) SLDA functionality consults with other entities in system to determine whether to grant or deny access privileges
6: CREATE/UPDATE/DELETE Response
Payload: Status: Success
7: (Optional) SLDA function updates <accessControlPolicy> privileges based on SLDA results
8: CREATE/RETRIEVE/UPDATE/DELETE/SUBSCRIBE/DISCOVER Request
To: URI of targeted resource
Payload: Representation of Resource (if applicable)
9: Entity checks <accessControlPolicy> resources of targeted resource or parent resource(s) and detects proper privileges exist and access is granted
10: CREATE/UPDATE/DELETE Response
Payload: Status: Success

Figure 7.2.2.2.1-1

- **Step 1:** A requesting oneM2M entity (e.g. AE or CSE) issues a request (i.e. Create/Retrieve/Update/Delete/Subscribe/Discover) to another oneM2M entity.

- **Step 2:** The receiving oneM2M entity (e.g. CSE) detects access to targeted resource. The receiving entity checks the corresponding <*accessControlPolicy*> resource(s) applicable to the targeted resource (if any) to determine whether the requester has sufficient privileges configured to allow it to perform the desired type of access. In this example, the requester lacks sufficient access privileges.

- **Step 3:** Rather than immediately returning an 'access denied' error to the requester, the receiving entity triggers dynamic authorization processing to be performed by the SLDA functionality that can either be hosted on the receiving entity or on another entity in the system. SLDA functionality can also be split across multiple entities with some functionality hosted on the receiving entity.

- **Step 4:** The receiving entity, if enabled with SLDA functionality, begins to perform dynamic authorization processing. Depending on whether the receiving entity hosts SLDA functionality itself will determine whether the SLDA processing is performed completely local or whether the receiving entity will communicate with other oneM2M entities in the system to assist it with performing SLDA. If no SLDA functionality is found the SLDA processing is stopped and an 'access denied' error is returned to the requester. If SLDA functionality is found to be hosted on either the receiving entity and/or other entities in the system, then this functionality is triggered to attempt to locate <*dynAuthPolicy*> resources applicable to the targeted resource. If a <*dynAuthPolicy*> is not found, the SLDA discontinues dynamic authorization processing and returns an 'access denied' error to the requester. If one or more <*dynAuthPolicy*> resources are found, the SLDA functionality uses the *dynAuthRules* attribute for each resource found to evaluate against the requester's request and determines whether privileges can be dynamically granted.

- **Step 5 (Optional):** While evaluating the *dynAuthRules* of the applicable <*dynAuthPolicy*> resources, the SLDA function can consult with other entities in the system depending on the type of *dynAuthRules* specified. Consultation can gather the proper information the SLDA function uses to determine whether or not to grant or deny access privileges to the requester. This consultation is performed using the <*consult*> resource and its corresponding request and response primitives.

- **Step 6:** Based on the results of evaluating the request against the *dynAuthRules* and possibly consulting with other entities in the system to obtain additional information that it factors into its decision making, the SLDA functionality decides to grant access to the requester. As a result, the receiving entity performs the request on the targeted resource and returns a successful response to the requester. The requester, is unaware that autonomous SLDA was performed. All it is aware of is its request completed successfully.

- **Step 7 (Optional):** Using the dynamic authorization results, the SLDA function can optionally update the *privileges* attribute within the <*accessControlPolicy*> resource(s) associated with the targeted resource to add access control privileges for the requester. This decision can be controlled via a *dynAuthRule* defined within a <dynAuthPolicy> resource.

EXAMPLE: The *dynAuthRule* can support a rule (e.g. privilege lifetime rule) to control whether or not the SLDA function adds or updates the static privileges of the <*accessControlPolicy*> resources and if so what the expiration time of this privilege is. As a result, the SLDA function can use this rule to control whether it updates or adds a privilege and if so the expiration time (e.g. this can be done using the *accessControlExpirationTime* component of the privilege). Based on these rules, the SLDA function determines whether or not to add/update privileges and their respective lifetimes. By choosing to update the privileges, the requester can be granted access to the resource and perform the same operation without dynamic authorization having to be repeated.

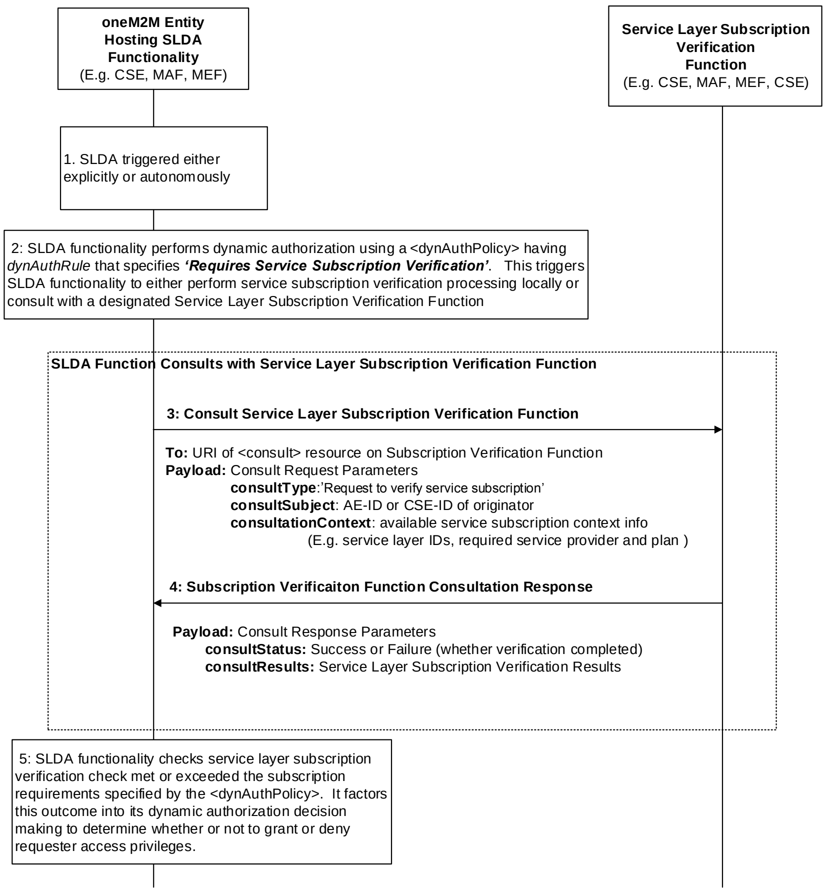
- **Step 8:** A requesting oneM2M entity (e.g. AE or CSE) issues a request (i.e. Create/Retrieve/Update/Delete/Subscribe/Discover) to the same resource on the same oneM2M entity.

- **Step 9:** The receiving oneM2M entity (e.g. CSE) detects access to a targeted resource. The receiving entity checks the corresponding <*accessControlPolicy*> resource(s) applicable to the targeted resource and detects that the requester now has sufficient access privileges as a result of the SLDA function having added these privileges as part of Step 7.

- **Step 10:** The receiving entity performs the request on the targeted resource and returns a successful response to the requester.

7.2.2.2.2 Dynamic Authorization Process - Based on Subscription

The SLDA can consult with an entity in the system (e.g. CSE, MAF, MEF) to determine whether or not a requesting entity has a proper service subscription that allows it privileges to gain access to a targeted resource.



oneM2M Entity Hosting SLDA Functionality (E.g. CSE, MAF, MEF)
Service Layer Subscription Verification Function (E.g. CSE, MAF, MEF, CSE)
1. SLDA triggered either explicitly or autonomously
2: SLDA functionality performs dynamic authorization using a <dynAuthPolicy> having dynAuthRule that specifies 'Requires Service Subscription Verification'. This triggers SLDA functionality to either perform service subscription verification processing locally or consult with a designated Service Layer Subscription Verification Function
SLDA Function Consults with Service Layer Subscription Verification Function
3: Consult Service Layer Subscription Verification Function
To: URI of <consult> resource on Subscription Verification Function
Payload: Consult Request Parameters
consultType:'Request to verify service subscription'
consultSubject: AE-ID or CSE-ID of originator
consultationContext: available service subscription context info (E.g. service layer IDs, required service provider and plan)
4: Subscription Verificaiton Function Consultation Response
Payload: Consult Response Parameters
consultStatus: Success or Failure (whether verification completed)
consultResults: Service Layer Subscription Verification Results
5: SLDA functionality checks service layer subscription verification check met or exceeded the subscription requirements specified by the <dynAuthPolicy>. It factors this outcome into its dynamic authorization decision making to determine whether or not to grant or deny requester access privileges.

Figure 7.2.2.2.2-1

- **Step 1:** The SLDA function is triggered either by an explicit request (i.e. access to <*dynAuthRequest*> resource) or autonomously via the SLDA function itself (e.g. via detection of a request that failed checks on existing <*accessControlPolicy*> static privileges).

- **Step 2:** The SLDA locates applicable <*dynAuthPolicy*> resources (if any) and evaluates the *dynAuthPolicy* attribute to determine what rules have been configured. In this example, the *dynAuthPolicy* attribute has been configured with a rule specifying **'Requires Service Subscription Verification'**. This rule triggers the SLDA function to either perform local service subscription verification processing (i.e. check <*m2mServiceSubscriptionProfile*> resources) or consult with a Subscription Verification Function in the system to have it perform this verification on behalf of the SLDA.

- **Step 3:** The SLDA function forms a consultation request message. The message is targeted to the <*consult*> resource hosted by the Subscription Verification Function. The *consultSubject* is configured with the AE-ID or CSE-ID of the requester for which the verification is being performed. The payload of the request is configured with any service subscription information of the requester that is available (e.g. service layer IDs) as well as the service subscription policy rule defining conditions such as the applicable service provider and/or service plan that a requester will be granted access privileges.

- **Step 4:** The Subscription Verification Function assesses the subscription conditions defined by the policy against the requester's service subscription to determine whether the requester meets the conditions defined by the policy. The Subscription Verification Function then returns the result back to the SLDA via a consult response containing a verification status and a list of conditions that have been met as well as any that have not.

- **Step 5:** SLDA functionality checks the subscription verification results and factors this outcome into its dynamic authorization decision making to determine whether or not to grant or deny requester access privileges.

7.2.3 Resource Definitions

7.2.3.1 Resource Type <*dynAuthPolicy*>

The <*dynAuthPolicy*> resource serves as the interface to configure the SLDA function with dynamic authorization rules. This resource is used to create, retrieve, update and delete SLDA policies. Once configured with one or more <*dynAuthPolicy*> resources, the SLDA function uses the rules defined within the policies to perform dynamic authorization decision making.

Various types of dynamic authorization rules can be supported such as (but not limited to) multi-factor authentication-based, payment-based, security assessment-based, consultation-based and reputation-based policies. The representation of the resources are described further below.

In addition to dynamic authorization rules, the <*dynAuthPolicy*> resource also supports a *selfAccessControlPolicyIDs* attribute. This attribute is used to reference access control policies that define the access privileges for the <*dynAuthPolicy*> resource itself. These privileges are used to control which entities are allowed to access (e.g. retrieve, update or delete) the <dynAuthPolicy> resource and its attributes.

Optionally, the <*dynAuthPolicy*> resource can also support an *accessControlPolicyIDs* attribute that can be configured with a list of links to one or more <accessControlPolicy> resources whose static privileges a SLDA function can dynamically create, update or delete using this <*dynAuthPolicy*> resource.

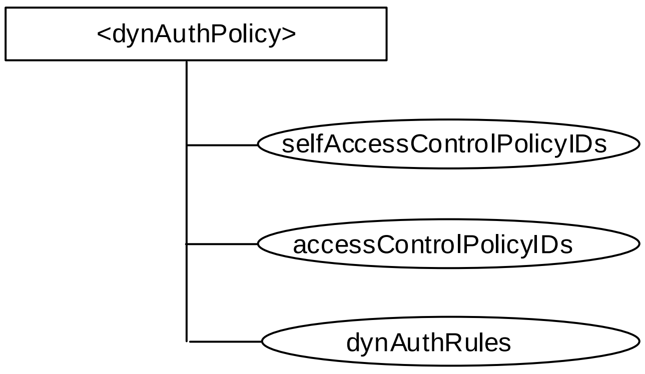


Figure 7.2.3.1-1: Structure of <*dynAuthPolicy*> resource

The <*dynAuthPolicy*> resource contains the attributes specified in table 7.2.3.1-1.

Table 7.2.3.1-1: Attributes of <*dynAuthPolicy*> resource

Attribute Name	Multiplicity	RW/RO/WO	Description
selfAccessControlPolicyIDs	1(L)	RO	Links to authorization policy(s) defining which entities have the privilege to perform CRUD operations to this <dynAuthPolicy> resource.
accessControlPolicyIDs	0..1(L)	RW	Links to one or more access control policies whose privileges can be dynamically created, updated or deleted by an SLDA function based on the rules defined by this <dynAuthPolicy> resource.
dynAuthRules	1(L)	RW	List of one or more rules used by a SLDA function to perform dynamic authorization. These rules are used to determine whether or not the SLDA function is to dynamically grant access privileges to an entity attempting to access a resource or service that the entity does not currently have proper privileges to access. Each rule can be specified as a list of criteria met by an originator of a request before access privileges are dynamically granted. The following is a list of types of criteria: (see note) • **Allowed Originators (ar):** A list of IDs of originators that are candidates for dynamic authorization. Wildcard characters such as a '*' can be used to. • **Blocked Originators (br):** A list of IDs of originators that are NOT candidates for dynamic authorization. Wildcard characters such as a '*' can be used to. • **Allowed Operations (aop):** A list of allowed operations that are candidates for dynamic authorization. • **Blocked Operations (bop):** A list of operations that are NOT candidates for dynamic authorization. • **Allowed Locations (al):** A list of locations originator is allowed to be in to be considered a candidate for dynamic authorization. • **Blocked Locations (bl):** A list of locations originator is forbidden to be in to be considered a candidate for dynamic authorization. • **Allowed Time Schedule (ats):** A list of times that requests are allowed to be considered a candidate for dynamic authorization. • **Blocked Time Schedule (bts):** A list of times that requests are NOT allowed to be considered a candidate for dynamic authorization. • **Allowed IP addresses (aia):** A list of originator IP addresses that are allowed to be considered candidates for dynamic authorization. • **Blocked IP addresses (aia):** A list of originator IP addresses that are NOT allowed to be considered candidates for dynamic authorization. • **Allowed Requester Roles (arr):** A list of requester roles that are allowed to be considered candidates for dynamic authorization. • **Blocked Requester Roles (brr):** A list of requester roles that are NOT candidates for dynamic authorization. • **Allowed Apps (aa):** A list of classes of apps or App-IDs that are allowed to be considered candidates for dynamic authorization. • **Blocked Apps (ba):** A list of classes of apps or App-IDs that are NOT candidates for dynamic authorization. • **Requires Payment Verification (rpv):** Originator's payment information is first verified before it can be considered a candidate for dynamic authorization. If a payment consultation contact is specified in the consultationContact list, then the SLDA function can consult with it.

Attribute Name	Multiplicity	RW/RO/WO	Description
			• **Requires Service Subscription Verification (rsv):** If present, then originator's service subscription first verifies to be valid before it can be considered a candidate for dynamic authorization. If a service subscription consultation contact is specified in the consultationContact list, then the SLDA function can consult with it. • **Requires Security Assessment (rsa):** If present, SLDA performs security assessment-based authorization to verify that originator meets a certain level of security. If an authorization consultation contact is specified in the consultationContact list, then the SLDA function can consult with it. • **Required Platform Validation Level (rpv):** A level of validation of an originator's platform to be considered a candidate for dynamic authorization. E.g. High (H), Average (A), Low (L). If a contact is specified in the consultationContact list for a platform validation function, the SLDA function can consult with it. • **Required Pre-Requisites (rpr):** A list of credentials, subscriptions, actions or other arrangements an originator establishes/completes to be considered a candidate for dynamic authorization. • **Requires Consultation-based Authorization (rca):** If present, SLDA performs consultation-based authorization. If an authorization consultation contact is specified in the consultationContact list, then the SLDA function can consult with it. • **Requires Payment Verification (rpv):** Originator's payment information is first verified before it can be considered a candidate for dynamic authorization. If a payment consultation contact is specified in the consultationContact list, then the SLDA function can consult with it. For example a list of dynAuthRules can be expressed as follows: • *ar:AE1,AE2,AE3; aop:U,R,S; boo:C,D; rsv.*
NOTE: Additional types can be defined.			

7.2.3.2 <*consult*> Resource

The <consult> resource is a virtual resource having no attributes. The URI of this resource functions as a consultation point of contact which an SLDA function uses when performing dynamic authorization to consult with another entity in the network. For example, when a <*dynAuthPolicy*> is configured with a consultation-based rule that defines a consultationContact, the URI of a <*consult*> resource is used. A <*consult*> resource can be hosted by various oneM2M defined entities such as CSEs, AEs, MEFs and MAFs. In addition, this resource can also be hosted by other types of entities not yet defined by oneM2M such as a location function, reputation verification function (e.g. social media servers), platform validation and assessment function, payment function, security function and service subscription verification function.

Figure 7.2.3.2-1: <*consult*> Virtual Resource

To perform consultation, an SLDA function constructs a consult request message and targets this message towards a <*consult*> resource hosted by an entity within the system. Upon receiving this request, the entity processes it according to the type of consult, constructs a corresponding response message with the consult results, and returns the response to the SLDA function.

Table 7.2.3.2-1: Parameters of Consult Request Message

Parameter Name	Description
consultantAddr	This is the address which this request is targeted at. For example, the URI of the <*consult*> resource.
consulterID	A oneM2M service layer ID of the originator issuing this consult request (E.g. AE-ID, CSE-ID, App-ID, etc). For example, the CSE-ID hosting an SLDA function.
consultType	The type of consult request. Several different types of consultation can be supported including one or more of the following types. • Request for Consultation-based Dynamic Authorization • Request for a Dynamic Authorization Credential • Request for the location of an specified entity • Request to verify payment info/token of a specified entity • Request to verify reputation of a specified entity • Request to verify service layer subscription of a specified entity • Request to validate platform entity is hosted on • Request to assess security of a specified entity
consultationInfo	Each type of consult request has corresponding information associated with it. This information can be sourced from an explicit dynamic authorization request or from an SLDA that is performing autonomous dynamic authorization. Some types of information include the following: • consultSubject - Identifier of an entity which is the subject being consulted on behalf of. For example, a oneM2M AE-ID for which an SLDA function is consulting on behalf of and issuing a location consult request to a location server to determine the AE's location. In this example, the AE-ID would be the consultSubject. • Security context/assessment information of the consultSubject which the SLDA can collect and track • Type of request • Requested privileges • Payment Info • Service Subscription Info • Security context information • Security assessment conditions

Table 7.2.3.2-2: Parameters of Consult Response Message

Parameter Name	Description
consultStatus	The status of the consult request: • Consultation Rejected • Consultation Successful • Consultation Error
consultResults	Each type of consult response can have corresponding results associated with it. Some of the results are generic and applicable to all the different types of consult requests, while some results are specific to a particular type as described below: • List of privileges granted to requested resource or service and corresponding lifetime of privileges • List of denied privileges to requested resource or service that were not granted and reasons why • List of privileges that were pro-actively granted to resources or services that were not requested but the requester might be interested in. • A dynamic authorization credential • A location of the requester • A payment verification status • Reputation reviews/ranking • Platform validation assessment results • Security assessment level (e.g. trusted, un-trusted, etc.)

7.2.3.3 Enhancements to <*accessControlPolicy*>

Current oneM2M specifications only support an <*accessControlPolicy*> resource having a set of defined *privileges* based on an ACL. This ACL is a set of access-control-rule tuples that can be stored within the *privileges* attribute, where each tuple is comprised of three components consisting of:

1) *accessControlOriginators*;
2) *accessControlOperations*; and
3) *accessControlContext*.

Three enhancements to the existing *privileges* attribute of the <*accessControlPolicy*> resource are described below:

1) An additional component has been added to privileges access-control-rule tuple called an *accessControlPrivilegeID*. This component can identify and address an individual privileges access-control-rule tuple. This identifier is useful when supporting updating or deleting of an individual tuple. Otherwise, an update or delete of an individual tuple results in updating the entire privileges access-control-rule tuple list which is less efficient.

2) Another component has been added to the privileges access-control-rule tuple called an *accessControlExpirationTime*. This component can define a lifetime associated with a particular access-control-rule tuple. In addition, it enables access-control-rule tuples to have different lifetimes with respect to one another, which can provide more flexibility from a privileges perspective.

3) Three additional types of *accessControlContext* have also been defined:
    - *accessControlPaymentTerms* - List of payment terms to be established before an entity is allowed to access resources associated with this authorization policy.
    - *accessControlReputationLevel* – Expected reputation level of an entity before it is allowed to access resources associated with this authorization policy
    - *accessControlSecurityAssessment* - Expected level of security of an entity before it is allowed to access resources associated with this authorization policy

7.3 Proposal 5: A Solution of Token Based Access Control

7.3.1 Token Structure

The structure of token is shown in figure 7.3.1-1, it contains the following data fields:

- version: version of the token format.
- tokenID: unique ID of the token.
- holder: ID of the token holder.
- issuer: ID of the token issuer.
- startTime: token valid from this time.
- expiryTime: token expired after this time.
- tokenType: it specifies what kind of privilege is specified in privileges field, e.g. a role or an Access Control List (ACL)..
- tokenName: human readable name of the token.
- appCategories: List of M2M application identities that specify in which applications this token can be used for access control.
- privileges: privilege description, e.g. roles or ACLs.

- extensions: information defined and used by a specific application.

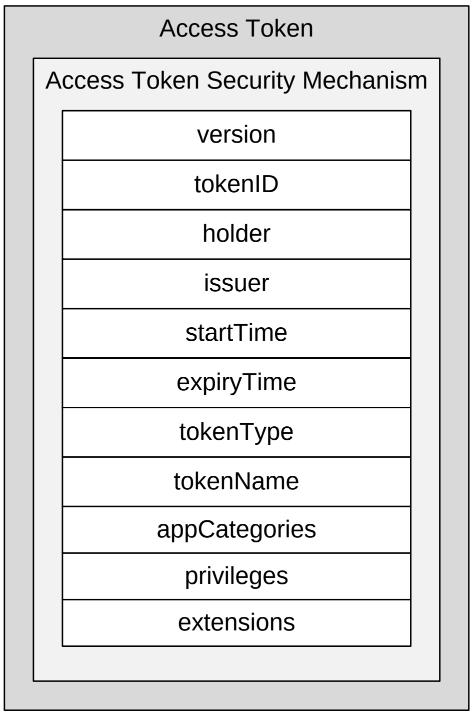


Figure 7.3.1-1: Structure of access token

7.3.2 Resource Type *token*

The <*token*> resource represents an access token assigned an AE or CSE. It is the child resource of an <*AE*> resource or a <*remoteCSE*> or directly under the <*CSEBase*> resource of the IN-CSE. One can get the information about what tokens are assigned to an AE or a CSE through retrieving <*token*> resources under its <*AE*> resource or <*remoteCSE*> resource or the <*CSEBase*> resource of the IN-CSE.

An <*AE*> resource or a <*remoteCES*> or the <*CSEBase*> resource of the IN-CSE can have zero or multiple <*token*> child resources.

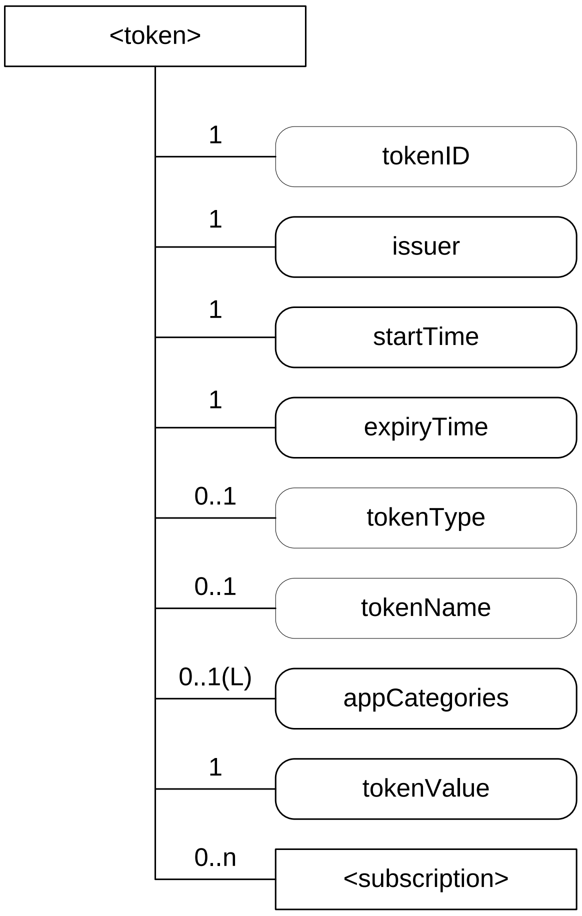


Figure 7.3.2-1: Structure of <token> resource

The <*token*> resource contains the child resources specified in table 7.3.2-1.

Table 7.3.2-1: Child resources of <*token*> resource

Child Resources of <*token*>	Child Resource Type	Multiplicity	Description	<*token*> Child Resource Types
[variable]	*<subscription>*	0..n	See clause 9.6.8 of oneM2M TS-0001 [i.2] where the type of this resource is described.	*[variable]*

The <*token*> resource contains the attributes specified in table 7.3.2-2.

Table 7.3.2-2: Attributes of <*token*> resource

Attributes of <*token*>	**Multiplicity**	**RW/ RO/ WO**	**Description**
resourceType	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
resourceID	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
resourceName	1	WO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
parentID	1	RO	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
expirationTime	1	RW	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described.
accessControlPolicyIDs	0..1 (L)	RW	See clause 9.6.1.3 of oneM2M TS-0001 [i.2] where this common attribute is described. If no *accessControlPolicyIDs* is given at the time of creation, the *accessControlPolicyIDs* of the parent resource is linked to this attribute.
creationTime	1	RO	See clause 9.6.1.3 where this common attribute is described.
labels	0..1 (L)	RW	See clause 9.6.1.3 where this common attribute is described
lastModifiedTime	1	RO	See clause 9.6.1.3 where this common attribute is described.
tokenID	1	RW	The identifier of the token.
issuer	1	RW	The identifier of the token issuer.
startTime	1	RW	Start time/date of the token can be used for access control. This attribute is mandatory for all token resources.
expiryTime	1	RW	End time/date of the token can be used for access control. This attribute is mandatory for all token resources.
tokenType	1	RW	Specify what kind of privileges is specified in the privileges field of token, e.g. roles or ACLs.
tokenName	0..1	RW	Human readable name of the token.
appCategories	0..1 (L)	RW	Specifies the M2M applications which can use this token for access control. The holder of the token can use this information to select applicable tokens for a resource access.
tokenValue	0..1	RW	Used to store access token itself. The token ID in the access token is expected to be equal to the value in the *tokenD* attribute.

7.3.3 Procedure of Token Based Access Control

The general procedure of issuing a token to an Originator and the originator uses the token to access resource is shown in the figure 7.3.3-1 and described as follows:

Pre-configuration for the token issuance and verification:

- The Originator registers to the Registrar CSE.
- The Token Authority provides the security credentials used for verifying tokens to PDPs.

Procedure of token issuance and use

1) The Originator sends token application request to the Token Authority. This step is not performed in some cases.

2) The Token Authority checks if the applied privilege can be assigned to the Originator. The Token Authority can contact an External Authorization Function for checking this request. In case Token Authority can check the privilege authorization locally, skip this step and the next step.

3) The External Authorization Function checks the privilege authorization request and returns the check result.

4) After passing the privilege authorization check the Token Authority issues a token that contains the applied privilege for the Originator.

5) The Token Authority sends a <token> resource creation request to the Originator's <*AE*>/<*remoteCSE*> resource which is in the Registrar CSE. The token issued to the AE/CSE and information pertain to the token are included in the request.

6) The Registrar CSE checks the access control policies to determine if the Token Authority has the privileges of creating <*token*> resource in the target <*AE*>/<*remoteCSE*> resource. If it is permitted, the Registrar CSE creates the <*token*> resource.

7) The Registrar CSE returns the result of <*token*> resource creation back to the Token Authority.

8) The Token Authority returns the result of token issuance back to the Originator. The response can contain the issued token and information pertaining to this token.

9) The Originator sends <*token*> resources retrieve request to its <*AE*>/<*remoteCSE*> resource in the Registrar CSE in order to get the tokens that are issued to it.

10) The Registrar CSE returns retrieved <*token*> resources back to the Originator.

11) The Originator selects applicable tokens according to the *appCategoryIDs* attribute of <*token*> resources, and includes them into the request sent to the Hosting CSE. The included token information contains at least the token IDs or tokens themselves.

12) The PEP in the Hosting CSE generates access decision request according to the request of the Originator, and sends the request to a PDP. The token information received from the Originator is included in the request.

13) In case only token IDs are included in the access decision request, the PDP sends a token attribute request to a PIP in order to get the token information using the token IDs, and the PIP further sends a <*token*> resource retrieve request to the Registrar CSE of the Originator. The PDP can directly send a <*token*> resource retrieve request to the Registrar CSE of the Originator instead of via a PIP. In case tokens are included in the access decision request, skip this step and the next step.

14) The Registrar CSE retrieves the <*token*> resource in the target <*AE*>/<*remoteCSE*> resource according to the token IDs, and returns the <*token*> resource back to the PIP or directly back to the PDP.

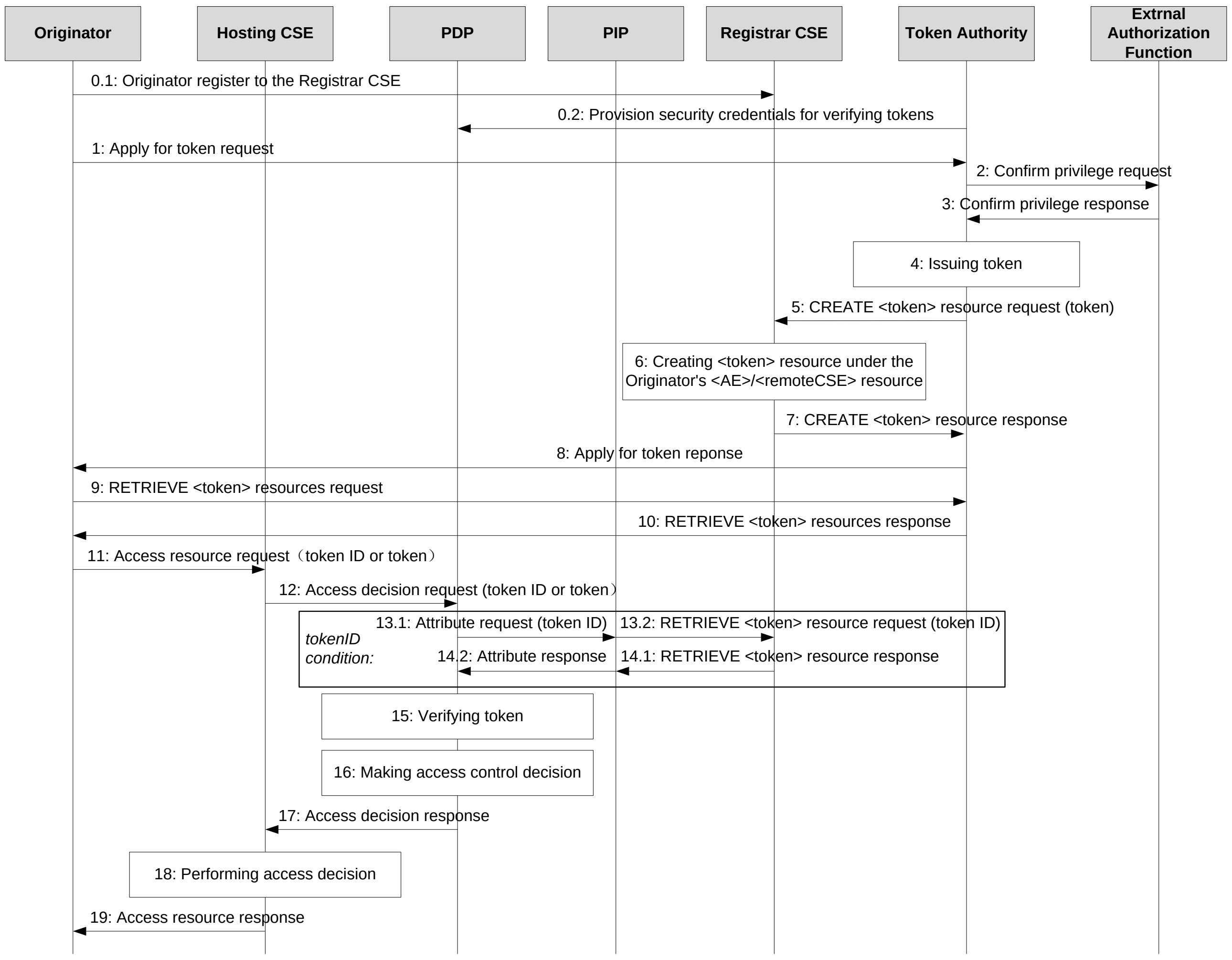
15) The PDP verifies the received tokens, the verification includes: a token is issued by a valid Token Authority and is still valid. If a token passes the verification, the PDP extracts the privileges from the token.

16) The PDP evaluates the resource access request of the Originator using access control policies and privileges assigned to the Originator for making an access control decision.

17) The PDP returns the access control decision back to the PEP.

18) The Hosting CSE enforces the access control decision, i.e. either performs the resource access on behalf the Originator or denies the resource access.

19) The Hosting CSE returns the result of resource access back to the Originator.



Originator
Hosting CSE
PDP
PIP
Registrar CSE
Token Authority
Extrnal Authorization Function
0.1: Originator register to the Registrar CSE
0.2: Provision security credentials for verifying tokens
1: Apply for token request
2: Confirm privilege request
3: Confirm privilege response
4: Issuing token
5: CREATE <token> resource request (token)
6: Creating <token> resource under the Originator's <AE>/<remoteCSE> resource
7: CREATE <token> resource response
8: Apply for token reponse
9: RETRIEVE <token> resources request
10: RETRIEVE <token> resources response
11: Access resource request（token ID or token）
12: Access decision request (token ID or token）
tokenID condition:
13.1: Attribute request (token ID)
13.2: RETRIEVE <token> resource request (token ID)
14.2: Attribute response
14.1: RETRIEVE <token> resource response
15: Verifying token
16: Making access control decision
17: Access decision response
18: Performing access decision
19: Access resource response

Figure 7.3.3-1: Token based access control procedure

8 Release 2 Dynamic Authorization

8.1 Overview of Release 2 Dynamic Authorization Features

The analysis in the preceding clauses was used to guide the specification of Release 2 features for Dynamic Authorization.

NOTE: This description of the Release 2 Dynamic Authorization features is largely copied from clause 11.5.1, oneM2M TS-0001 [i.2].

The Dynamic Authorization reference model for Release 2 is shown in figure 8.1-1.

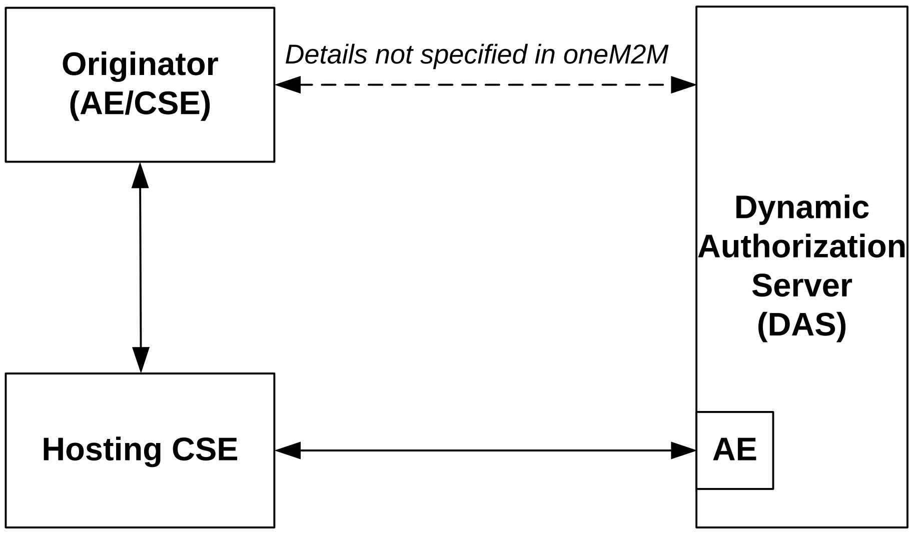

Originator (AE/CSE)
Details not specified in oneM2M
Dynamic Authorization Server (DAS)
Hosting CSE
AE

Figure 8.1-1: Dynamic Authorization reference model

This reference model added a new entity: a Dynamic Authorization Server (DAS). The DAS is a server configured with policies for dynamic authorization, and provided with credentials for issuing Tokens.

This reference model is simpler than most of the reference models proposed in clause 6. The oneM2M Security Working group decided to start with a reference model with a flexible DAS which does not specify the protocols for interaction with the Originator and does not specify the procedures by which the DAS decides whether or not to provide an Originator with authorization. The DAS can include an AE for specified interaction with the oneM2M system.

The following procedures are specified:

- **Direct Dynamic Authorization**, summarized in figure 8.1-2: In this procedure, Hosting CSE interacts with the DAS to obtain Dynamic Authorization. This procedure is based on the proposal Dynamic Authorization Architecture Proposal 4 (DAA4) in clause 6.5 and Service Layer Dynamic Authorization (SLDA) in clause 7.2.

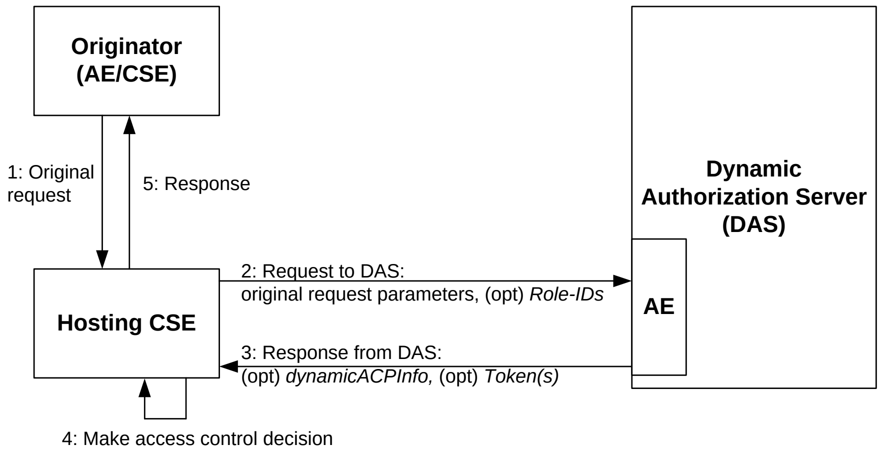

Originator (AE/CSE)
1: Original request
5: Response
Dynamic Authorization Server (DAS)
2: Request to DAS: original request parameters, (opt) Role-IDs
Hosting CSE
AE
3: Response from DAS: (opt) dynamicACPInfo, (opt) Token(s)
4: Make access control decision

Figure 8.1-2: Direct Dynamic Authorization

- **Indirect Dynamic Authorization**, summarized in figure 8.1-3: Indirect Dynamic Authorization provides a mechanism to trigger the Originator to request one or more Tokens from one or more DAS. This mechanism can provide an advantage over Direct Dynamic Authorization in cases where the DAS performs some additional authentication and authorization of the Originator.

  EXAMPLE: The DAS policies can initiate authentication of the current user of the Originator, and/or explicit authorization by the current user.

- Steps 1-2: The Hosting CSE can provide the Originator with ***Token Request Information*** in the unsuccessful response.
- Steps 3: The Originator interacts with the DAS with the intention that the DAS issue *Tokens* authorizing the Originator, and the Originator is provided with the Token or a Token-ID. The interaction is not described in the present specification.
- Steps 4-7: The Originator provides the Hosting CSE with a *Token, Token-ID* to indicate that the Token is to be considered in the access decision. In the case of a Token-ID, the Hosting CSE retrieves the corresponding Token via an AE of the DAS. These are then used in the access decision. The Hosting CSE can provide the Originator with a more compact *Local-Token-ID* which can be used to identify the Token to the Hosting CSE.

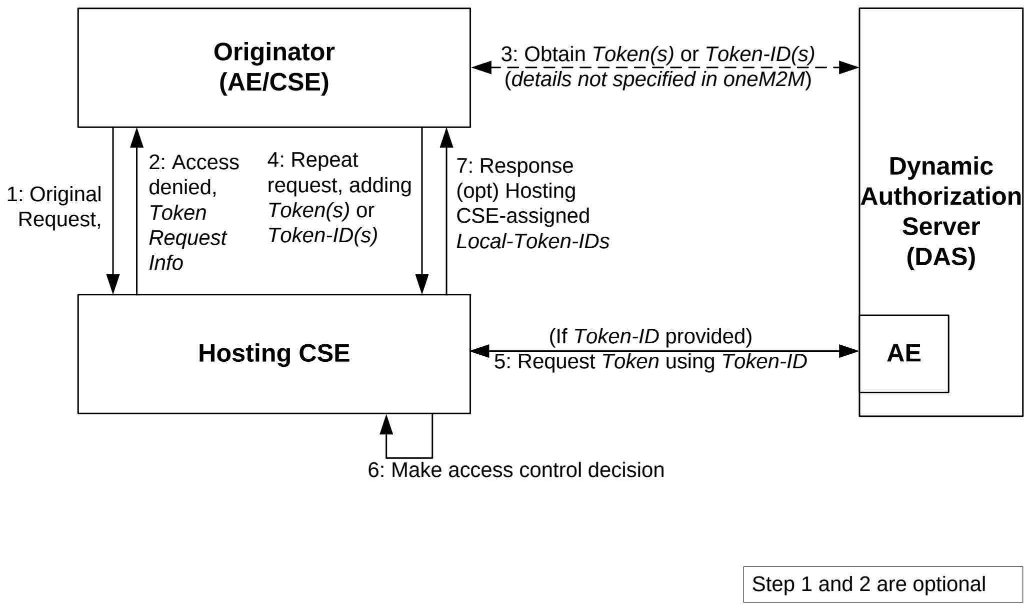

Originator (AE/CSE)
3: Obtain Token(s) or Token-ID(s) (details not specified in oneM2M)
1: Original Request,
2: Access denied, Token Request Info
4: Repeat request, adding Token(s) or Token-ID(s)
7: Response (opt) Hosting CSE-assigned Local-Token-IDs
Dynamic Authorization Server (DAS)
Hosting CSE
(If Token-ID provided) 5: Request Token using Token-ID
AE
6: Make access control decision
Step 1 and 2 are optional

Figure 8.1-3: Indirect Dynamic Authorization

Release 2 also included a Role-Based Access Control (RBAC) architecture in clause 7.4 of oneM2M TS-0003 [i.3]. This RBAC architecture includes the ability to issue Tokens associated with a Role assigned to the Originator, although the token issuing is less dynamic than the indirect Dynamic Authorization described above. The RBAC architecture also supports storing Tokens, in a discoverable form, on CSEs acting as Token Repositories, borrowing ideas from DAA5 in clause 6.6 and Proposal 1 in clause 7.1. The Release 2 RBAC architecture is not discussed further in this document.

8.2 Overview of Direct Dynamic Authorization

Direct Dynamic Authorization is based on proposal Dynamic Authorization Architecture Proposal 4 (DAA4) in clause 6.5. Direct Dynamic Authorization in Release 2 does not support use of a Token Repository (see final paragraph of clause 8.1), which was based on the use of the Registrar CSE in DAA5 in clause 6.6. Support for Token Repositories can be added in later releases.

The <*dynamicAuthorizationConsultation*> resource is defined in oneM2M TS-0001 [i.2] to describe how a Hosting CSE can submit requests to a DAS. If a resource includes a link to this resource, then access to that resource is allowed to be authorized by submitting requests to the DAS.

An interface is specified for a Direct Dynamic Authorization interactions between a Hosting CSE and a DAS. The parameters exchanged, and the corresponding processing, are specified in clause 7.3.2.2, oneM2M TS-0003 [i.3]. The transportation of parameters when oneM2M primitives are used is specified in clause 11.5.2, oneM2M TS-0001 [i.2].

The Hosting CSE to send, to the DAS, a DAS Request including:

- Parameters of the request from the Originator,
- A list of roles which might, if assigned to the Originator, permit the request based on existing ACPs.

The DAS has multiple options for authorizing access:

- The DAS can update the applicable access control rules using existing mechanisms for managing access control rules, if permitted by the Hosting CSE. This can include updating existing <*accessControlPolicy*> resources, creating new <*accessControlPolicy*> resources, and updating the *accessControlPolicyIDs* attribute of the requested resource.
- The DAS returns a DAS Response which can include:
  - A dynamic <*accessControlPolicy*> resource to be linked to the requested resource,
  - Permission to access the specific request,
  - Assignment of one or more Roles to the Originator,
  - Tokens identified in the Originator's request, and
  - Additional Tokens issued at the discretion of the DAS.

The DAS Request is sent in the securityInfo element in the notification data of a NOTIFY Request. The DAS Response is sent in the securityInfo element in the notification data of the corresponding NOTIFY Response.

8.3 Overview of Indirect Dynamic Authorization

Indirect Dynamic Authorization borrows ideas from the Dynamic Authorization Architecture Proposals DAA1, DAA2, DAA3 and DAA5 in clauses 6.2, 6.3, 6.4 and 6.6. The reference model is simpler than those proposals, primarily due to the decision to not specify the interface between Originator and DAS. This resulted in removing the separation of Grant Issuer and Grant Approver in DAA1, removing the Security Function and Token Authorization Function in DAA2, removing the Requesting Authority in DAA3 and removing the External Authorization Function in DAA5. Indirect Dynamic Authorization in Release 2 does not support use of a Token Repository (see final paragraph of clause 8.1), which was based on the use of the Registrar CSE in DAA5 in clause 6.6. Support for Token Repositories can be added in later releases.

The parameters exchanged, and the corresponding processing, are specified in clause 7.3.2.3 of oneM2M TS-0003 [i.3]. The transportation of parameters when oneM2M primitives are used is specified in clause 11.5.3 of oneM2M TS-0001 [i.2].

NOTE: This description of the Release 2 Indirect Dynamic Authorization features is largely copied from clause 11.5.3 of oneM2M TS-0001 [i.2].

The following description expands on the description of figure 8.1-3.

1. (Optional) The Originator sends a request to the Hosting CSE. This request can include Tokens, Token IDs or Local Token IDs, but this message flow assumes that these do not provide sufficient permissions for accessing the requested resource.
2. (Optional) Initial Hosting CSE processing:

   2.1. The Hosting CSE performs the access decision. If access is allowed, then Indirect Dynamic Authorization is not performed. If access is denied, then Indirect Dynamic Authorization can be performed.

   2.2. The Hosting CSE forms a ***Token Request Information*** primitive parameter containing DAS Requests for one or more DAS, which the Originator can forward to the corresponding DASs to request indirect dynamic authorization. The DAS Requests are formed in the same manner as for Direct Dynamic Authorization. Each DAS Requests can be encrypted and/or signed by the Hosting CSE using End-to-End Security of Data (ESData), for decryption and/or verification by the DAS.

   2.3. The Hosting CSE's response includes the ***Response Status Code*** "ORIGINATOR_HAS_NO_PRIVILEGE" and ***Token Request Information***.
3. The Originator selects a DAS identified in ***Token Request Information*** primitive parameter. The Originator interacts with the DAS to request the issuance of one or more Tokens. The Originator can provide information for the DAS provided in the ***Token Request Information***, and parameters from the original resource access request. The DAS issues a Token(s) and provides the Token-ID(s) and optionally the ESData-protected Token(s) to the

Originator. The DAS can also provide the Originator with other parameters from the Token; for example, the time window in which the Token is valid. This interaction is specific to the Dynamic Authorization System technology being used.

The DAS can also update the applicable access control rules using existing mechanisms for managing access control rules, as discussed in clause 8.2.

4. The Originator repeats the original resource access request, adding ESData-protected Token(s) provided by the DAS, and/or Token-ID(s) if the ESData-protected Token(s) was not provided by the DAS.

5. (Optional) If the request includes Token-ID(s), then for each Token-ID the Hosting CSE identifies the corresponding DAS from which to request the corresponding ESData-protected Token. The Hosting CSE requests the ESData-protected Token(s) from the DAS using the procedure defined for Direct Dynamic Authorization defined in clause 8.2.

6. Hosting CSE Processing:

   6.1. The Hosting CSE processes ESData-protected Token(s) to extract the authenticated Token(s). Additional checking is also applied. The Hosting CSE can cache the Token(s).

   6.2. The Hosting CSE can assign Local-Token-ID(s) to cached Token(s). This is an optimization allowing the Originator to provide this short Local-Token-ID rather than the globally-unique Token-ID or full Token.

   6.3. The Hosting CSE performs the access decision, including the authorizations provided in the authenticated Token(s), and performs the requested operation if allowed.

7. Response:

   7.1. For each new Local-Token-ID(s) that has been assigned, the Local-Token-ID and corresponding Token-ID are included in the ***Assigned Token Identifiers*** parameter of the response.

   7.2. In subsequent requests, the Originator can use the Token-ID or Token or Local-Token-ID (if provided).

8.4 Overview of Tokens

The Token structure is specified in clause 7.3.2.4, oneM2M TS-0003 [i.3]. The data fields, shown in table 8.4-1, are based on the proposal in clause 7.3.1.

Table 8.4-1: Data fields of a Token

Element	Multiplicity	Description
version	1	See Clause 8.4.1
tokenID	1	See Clause 8.4.1. This ID is globally unique within the lifetime of the Token
holder	1	See Clause 8.4.1
issuer	1	See Clause 8.4.1
notBefore	1	See startTime in Clause 8.4.1
notAfter	1	See expiryTime in Clause 8.4.1
tokenName	0..1	See Clause 8.4.1
audience	0..1	Defined by [i.10]. Identifies the Hosting CSE that the Token intended for
permissions	0..1 (L)	Describes the roles assigned by the Token, access control rules
extension	0..1	

The audience element is added to the proposal in clause 8.4.1, intended to limit the scope of Hosting CSEs with which the Token can be used.

The appCategories and privileges in clause 8.4.1 become child elements of the permissions element, with structure described in the table 8.4-2.

Table 8.4-2: Structure of the token permission

Element	Multiplicity	Description
resourceIDs	0..1	The resources to which this permission applies. If the privileges element is present, then this element is present.
privileges	0..1	A set of access control rules applicable to the identified resources (for the identified holder)
roleIDs	0..1	A set of role IDs applicable to the identified resources (for the identified holder)

The resourceIDs element is an addition to the proposal in clause 8.4.1, intended to limit the scope of resources where the Token can be used to obtain access. The privileges element was already in clause 8.4.1, while roleIDs is considered a more appropriate replacement for the appCategories in clause 8.4.1

The protocol-level structure of a Token is specified by the m2m:tokenClaimSet complex data type in oneM2M TS-0004 [i.9].

Clause 7.3.2.6 in oneM2M TS-0003, Release 2 [i.3] specifies a profile of JSON Web Tokens (JWT) in IETF RFC 7519 [i.10] for representing Tokens used in oneM2M. A JWT can be signed and/or encrypted by the DAS, for verification and/or decryption by the Hosting CSE. The payload comprises a set of JWT claims, with IETF RFC 7519 [i.10] standardizing an initial set of JWT claim names. IANA maintains a registry of JWT claim names [i.11]. Table 7.3.2.6.2-1, oneM2M TS-0003 [i.3] provides the mapping from the Token data fields in table 8.4-1 to the JWT claim names in a oneM2M-compliant JWT.

The RBAC architecture in clause 7.4 of oneM2M TS-0003 [i.3] defines a Token Repository which can store oneM2M Tokens in <*token*> resources, with the JWT claims copied to <*token*> resource attributes. Tokens stored in <*token*> resources can be discovered using the ***Filter Criteria*** request parameter.

Token evaluation is specified in clause 7.3.2.5, oneM2M TS-0003 [i.3].

9 Conclusion

The present document offers an overview of the use cases, requirements, architecture proposals and available solutions for Dynamic Authorization.

Some of the contents have been standardized as Release 2 Technical Specification, as described in clause 8. Others can be used to facilitate future normative work resulting in oneM2M Technical Specifications.

History

This clause shall be the last one in the document and list the main phases (all additional information will be removed at the publication stage).

Publication history		
V.2.0.0	07 December 2016	Approved for publication at TP#26