INTERNATIONAL STANDARD

ISO/IEC 30118-2

First edition
2018-11

Information technology — Open Connectivity Foundation (OCF) Specification —

Part 2: **Security specification**

Technologies de l'information — Spécification de la Fondation pour la connectivité ouverte (Fondation OCF) —

Partie 2: Spécification de sécurité

Reference number
ISO/IEC 30118-2:2018(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by the Open Connectivity Foundation (OCF) (as the OCF Security Specification, Version 1.0.0) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

A list of all parts in the ISO/IEC 30118 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

CONTENTS

Figures

Tables

1 Scope

This specification defines security objectives, philosophy, resources and mechanism that impacts OCF base layers of the OCF Core Specification. The OCF Core Specification contains informative security content. The OCF Security specification contains security normative content and may contain informative content related to the OCF base or other OCF specifications.

2 Normative References

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

OCF Core Specification, version 1.1, Open Connectivity Foundation, October 11, 2016. Latest version available at: https://openconnectivity.org/specs/OCF_Core_Specification.pdf.

OCF Device Specification, version 1.1, Open Connectivity Foundation, October 11, 2016. Latest version available at: https://openconnectivity.org/specs/OCF_Device_Specification.pdf.

OCF Resource Type Specification, version 1.1, Open Connectivity Foundation, October 11, 2016. Latest version available at: https://openconnectivity.org/specs/OCF_Resource_Type_Specification.pdf.

JSON SCHEMA, draft version 4, JSON Schema defines the media type "application/schema+json", a JSON based format for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON Available at: http://json-schema.org/latest/json-schema-core.html.

RAML, Restful API modelling language version 0.8. Available at: http://raml.org/spec.html.

OCF Security Resource Definitions, *API Definition Language for OCF Security Components,* Release OCF-v1.0.0
https://github.com/openconnectivityfoundation/security

3 Terms, Definitions, Symbols and Abbreviations

Terms, definitions, symbols and abbreviations used in this specification are defined by the OCF Core Specification. Terms specific to normative security mechanism are defined in this document in context.

This section restates terminology that is defined elsewhere, in this document or in other OCF specifications as a convenience for the reader. It is considered non-normative.

3.1 Terms and definitions

3.1.1
Access Manager Service

The Access Manager Service dynamically constructs ACL Resources in response to a Device Resource request. An Access Manager Service can evaluate access policies remotely and supply the result to a Server which allows or denies a pending access request.

3.1.2
ACL Provisioning Service

A name and Resource Type (oic.sec.aps) given to a Device that is authorized to provision ACL Resources.

3.1.3
Bootstrap Service

A Device that implements a service of type oic.sec.bss

3.1.4
Client

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.5
Credential Management Service

A name and Resource Type (oic.sec.cms) given to a Device that is authorized to provision credential Resources.

3.1.6
Device

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.7
Device Class

As defined in RFC 7228. RFC 7228 defines classes of constrained devices that distinguish when the OCF small footprint stack is used vs. a large footprint stack. Class 2 and below is for small footprint stacks.

3.1.8
Device ID

A stack instance identifier.

3.1.9
Entity

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.10
Interface

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.11
Intermediary

A Device that implements both Client and Server roles and may perform protocol translation, virtual device to physical device mapping or Resource translation

3.1.12
OCF Cipher Suite

A set of algorithms and parameters that define the cryptographic functionality of a Device. The OCF Cipher Suite includes the definition of the public key group operations, signatures, and specific hashing and encoding used to support the public key.

3.1.13
Onboarding Tool

A logical entity within a specific IoT network that establishes ownership for a specific device and helps bring the device into operational state within that network

3.1.14
Out of Band Method

Any mechanism for delivery of a secret from one party to another, not specified by OCF

3.1.15
Owner Credential

Credential, provisioned by an Onboarding Tool to a Device during onboarding, for the purposes of mutual authentication of the Device and Onboarding Tool during subsequent interactions

3.1.16
Platform ID

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.17
Property

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.18
Resource

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.19
Role (network context)

Stereotyped behavior of a Device; one of [Client, Server or Intermediary]

3.1.20
Role (Security context)

A Property of an OCF credentials Resource that names a role that a Device may assert when attempting access to Device Resources. Access policies may differ for Client if access is attempted through a role vs. the device UUID. This document assumes the security context unless otherwise stated.

3.1.21
Secure Resource Manager

A module in the OCF Core that implements security functionality that includes management of security Resources such as ACLs, credentials and Device owner transfer state.

3.1.22
Security Virtual Resource

An SVR is a resource supporting security features.

3.1.23
Server

Note 1 to entry: The details are defined in OCF Core Specification.

3.1.24
Trust Anchor

A well-defined, shared authority, within a trust hierarchy, by which two cryptographic entities (e.g. a Device and an onboarding tool) can assume trust

3.1.25
Unique Authenticable Identifier

A unique identifier created from the hash of a public key and associated OCF Cipher Suite that is used to create the DeviceID. The ownership of a UAID may be authenticated by peer Devices.

3.2 Symbols and Abbreviations

Symbol	Description
ACE	Access Control Entry
ACL	Access Control List
AMS	Access Manager Service
APS	ACL Provisioning Service
BSS	Bootstrap Service
CMS	Credential Management Service
CRUDN	CREATE, RETREIVE, UPDATE, DELETE, NOTIFY
CSR	Certificate Signing Request
ECDSA	Elliptic Curve Digital Signature Algorithm
EPC	Embedded Platform Credential
DPKP	Dynamic Public Key Pair
OC	Owner Credential
OCSP	Online Certificate Status Protocol
OBT	Onboarding Tool
PIN	Personal Identification Number
PSI	Persistent Storage Interface
RNG	Random Number Generator
SACL	Signed Access Control List
SE	Secure Element
SRM	Secure Resource Manager
SVR	Security Virtual Resource
TEE	Trusted Execution Environment
UAID	Unique Authenticable Identifier

Table 1 – Symbols and abbreviations

3.3 Conventions

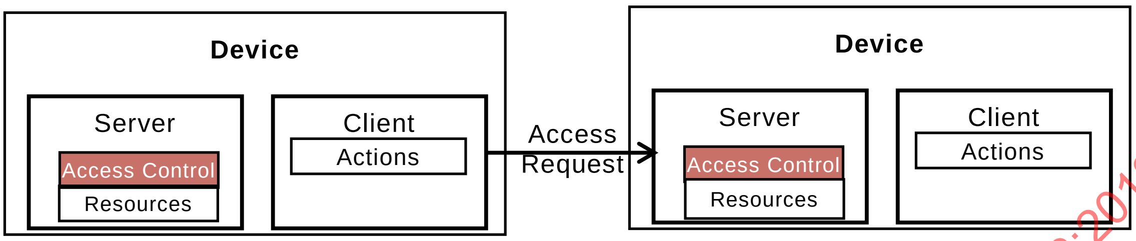


Figure 1 – OCF Interaction

Devices may implement a Client role that performs Actions on Servers. Actions access Resources managed by Servers. The OCF stack enforces access policies on Resources. End-to-end Device interaction can be protected using session protection protocol (e.g. DTLS) or with data encryption methods.

4 Document Conventions and Organization

This document defines Resources, protocols and conventions used to implement security for OCF core framework and applications.

For the purposes of this document, the terms and definitions given in OCF Core Specification apply.

4.1 Notation

In this document, features are described as required, recommended, allowed or DEPRECATED as follows:

Required (or **shall** or **mandatory**).

These basic features shall be implemented to comply with OCF Core Architecture. The phrases "shall not", and "PROHIBITED" indicate behavior that is prohibited, i.e. that if performed means the implementation is not in compliance.

Recommended (or **should**).

These features add functionality supported by OCF Core Architecture and should be implemented. Recommended features take advantage of the capabilities OCF Core Architecture, usually without imposing major increase of complexity. Notice that for compliance testing, if a recommended feature is implemented, it shall meet the specified requirements to be in compliance with these guidelines. Some recommended features could become requirements in the future. The phrase "should not" indicates behavior that is permitted but not recommended.

Allowed (or allowed).

These features are neither required nor recommended by OCF Core Architecture, but if the feature is implemented, it shall meet the specified requirements to be in compliance with these guidelines.

Conditionally allowed (CA)

The definition or behaviour depends on a condition. If the specified condition is met, then the definition or behaviour is allowed, otherwise it is not allowed.

Conditionally required (CR)

The definition or behaviour depends on a condition. If the specified condition is met, then the definition or behaviour is required. Otherwise the definition or behaviour is allowed as default unless specifically defined as not allowed.

DEPRECATED

Although these features are still described in this specification, they should not be implemented except for backward compatibility. The occurrence of a deprecated feature during operation of an implementation compliant with the current specification has no effect on the implementation's operation and does not produce any error conditions. Backward compatibility may require that a feature is implemented and functions as specified but it shall never be used by implementations compliant with this specification.

Strings that are to be taken literally are enclosed in "double quotes".

Words that are emphasized are printed in *italic*.

4.2 Data types

See OCF Core Specification.

4.3 Document structure

Informative sections may be found in the Overview sections, while normative sections fall outside of those sections.

The Security specification may use RAML as a specification language and JSON Schemas as payload definitions for all CRUDN actions. The mapping of the CRUDN actions is specified in the OCF Core Specification.

5 Security Overview

This is an informative section. The goal for the OCF security architecture is to protect the Resources and all aspects of HW and SW that are used to support the protection of Resource. From OCF perspective, a Device is a logical entity that conforms to the OCF specifications. In an interaction between the Devices, the Device acting as the Server holds and controls the Resources and provides the Device acting as a Client with access to those Resources, subject to a set of security mechanisms. The Platform, hosting the Device may provide security hardening that will be required for ensuring robustness of the variety of operations described in this specification.

The security theory of operation is described in the following steps.

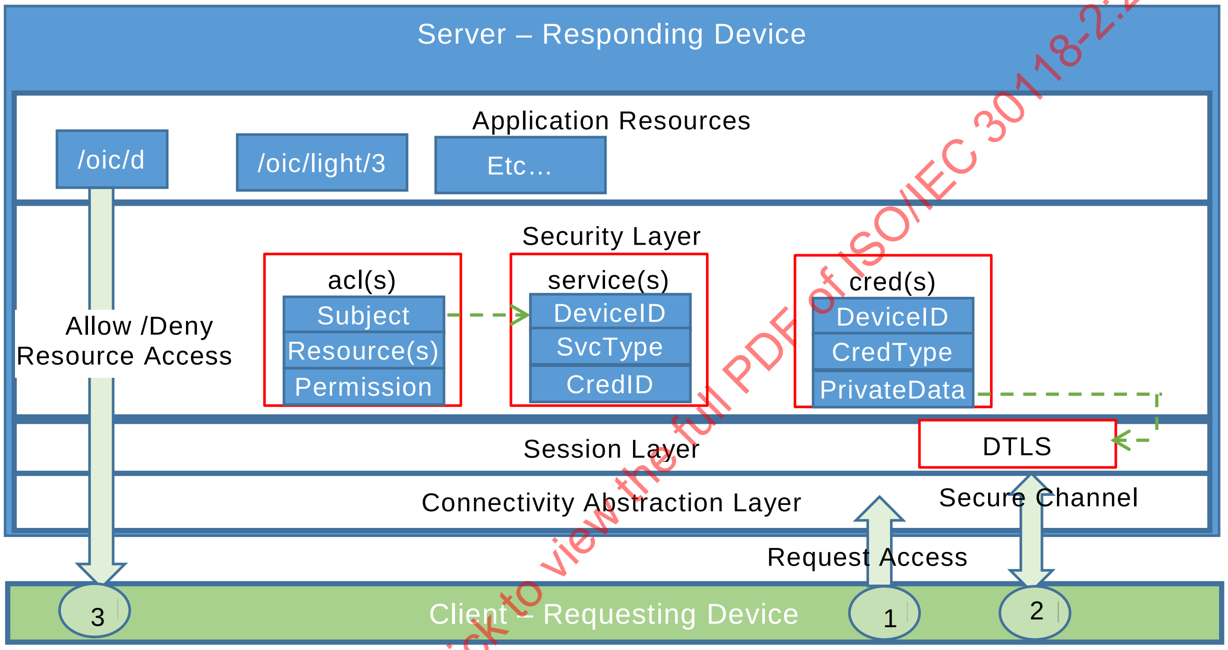


Figure 2 – OCF Layers

1. The Client establishes a network connection to the Server (Device holding the Resources). The connectivity abstraction layer ensures the Devices are able to connect despite differences in connectivity options.
2. The Devices (e.g. Server and Client) exchange messages either with or without a mutually-authenticated secure channel between the two Devices.
   - The oic.sec.cred Resource on each Devices holds the credentials used for mutual authentication and (when applicable) certificate validation.
   - Messages received over a secured channel are associated with a deviceUUID. In the case of a certificate credential, the deviceUUID is in the certificate received from the other Device. In the case of a symmetric key credential, the deviceUUID is configured with the credential in the oic.sec.cred Resource. There should be a binding between the device context and the Platform implementing the Device.
   - The Server can associate the Client with any number of allowed roleid. In the case of mutual authentication using a certificate, the allowed roleid (if any) are provided in role certificates; these are configured by the Client to the Server. In the case of a symmetric key, the allowed roleid (if any) are configured with the credential in the oic.sec.cred.
   - Requests received by a Server over an unsecured channel are treated as anonymous and not associated with any deviceUUID or roleid.
3. The Client submits a request to the Server.

- If the request is to be sent over the secure channel, then the Client can either explicitly assert specific roleid by including 'role' options in the request, or implicitly assert all roleid associated with the Client by including no 'role' options.

4. The Server receives the request.

   a. If the request is received over an unsecured channel, the Server treats the request as anonymous and no deviceUUID or roleid are associated with the request.
   b. If the request is received over a secure channel, then the Server associates the deviceUUID, and the Server either validates any explicitly asserted roleids by matching to an allowed roleid of the Client, or implicitly asserts all valid roleid of the Client.
   c. The Server then consults the Access Control List (ACL), and looks for an ACL entry matching the following criteria:
      - The requested Resource matches a Resource reference in the ACE
      - The requested operation is allowed by the "permissions" of the ACE, and
      - The "subjectUUID" contains either a special wildcard value matching all Devices or, if the Device is not anonymous, the subject matches the Client Deviceid or a valid asserted roleID. In certain cases, the requester may assert a role, if privileged access is required.

      If there is a matching ACE, then access to the Resource is allowed; otherwise access is denied. Access is enforced by the Server’s Secure Resource manager (SRM).

Resource protection includes protection of data both while at rest and during transit. It should be noted that, aside from access control mechanisms, OCF security specification does not include specification of secure storage of Resources, while stored at Servers. However, at rest protection for security Resources is expected to be provided through a combination of secure storage and access control. Secure storage can be accomplished through use of hardware security or encryption of data at rest. The exact implementation of secure storage is subject to a set of hardening requirements that are specified in Section 15 and may be subject to certification guidelines.

Data in transit protection, on the other hand, will be specified fully as a normative part of this specification. In transit protection may be afforded at

1. Resource layer through mechanisms such as JSON Web Encryption (JWE) and JSON Web Signatures (JWS) that allow payload protection independent of underlying transport security. This may be a necessary for transport mechanisms that cannot take advantage of DTLS for payload protection.

2. At transport layer through use of mechanisms such as DTLS. It should be noted that DTLS will provide packet by packet protection, rather than protection for the payload as whole. For instance, if the integrity of the entire payload as a whole is required, separate signature mechanisms must have already been in place before passing the packet down to the transport layer.

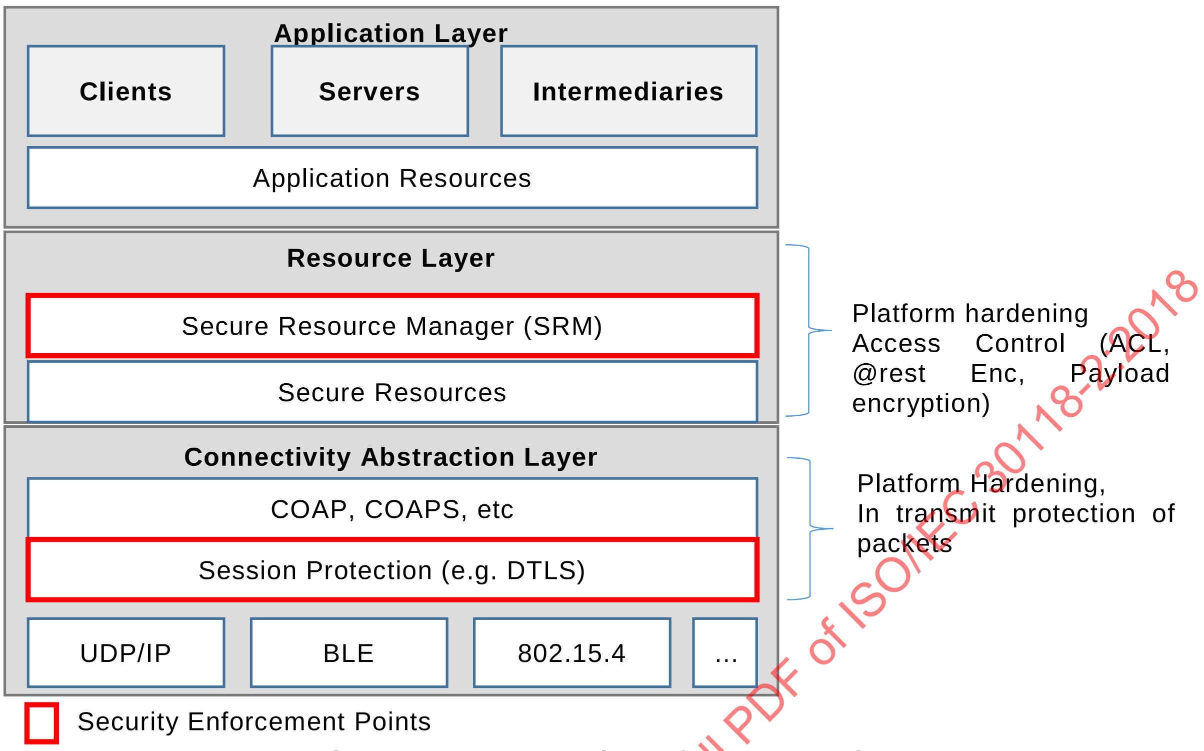

Application Layer
Clients
Servers
Intermediaries
Application Resources
Resource Layer
Secure Resource Manager (SRM)
Secure Resources
Platform hardening Access Control (ACL, @rest Enc, Payload encryption)
Connectivity Abstraction Layer
COAP, COAPS, etc
Session Protection (e.g. DTLS)
Platform Hardening, In transmit protection of packets
UDP/IP
BLE
802.15.4
...
Security Enforcement Points

Figure 3 – OCF Security Enforcement Points

5.1 Access Control

The OCF framework assumes that Resources are hosted by a Server and are made available to Clients subject to access control and authorization mechanisms. The Resources at the end point are protected through implementation of access control, authentication and confidentiality protection. This section provide an overview of Access Control (AC) through the use of ACLs However, AC in the OCF stack is expected to be transport and connectivity abstraction layer agnostic.

Implementation of access control relies on a-priori definition of a set of access policies for the Resource. The policies may be stored by a local ACL or an Access Manager Service (AMS) in form of Access Control Entries (ACE), where each ACE defines permissions required to access a specific Resource along with an optional validity period for the granted permission. Two types of access control mechanisms can be applied:

- Subject-based access control (SBAC), where each ACE will match a subject (e.g. identity of requestor) of the requesting entity against the subject included in the policy defined for Resource. Asserting the identity of the requestor requires an authentication process.
- Role-based Access Control (RBAC), where each ACE will match a role required by policy for the Resource to a role taken by the entity requesting access. Asserting the role of the requestor requires proper authorization.

In the OCF access control model, each Resource instance is required to have an associated access control policy. This means, each Device acting as Server, needs to have an ACL for each Resource it is protecting. Lack of an ACE that matches, it results in the Resource being inaccessible.

The ACE must match both the subject (i.e. OCF Client) and the Resource requested for the ACE to apply. There are multiple ways a subject could be matched, (1) device id, (2) role or (3) wildcard. The way in which the client connects to the server may be relevant context for making access control decisions. Wildcard matching on authenticated vs. unauthenticated and encrypted vs. unencrypted connection allows an access policy to be broadly applied to subject classes.

Example Wildcard Matching Policy:

```
"aclist2": [
  {
      "subject": {"conntype" : "anon-clear" },

      "resources":[
          { "wc":"*" }
        ],
      "permission": 31
  },
  {
      "subject": {"conntype" : "auth-crypt" },

      "resources":[
          { "wc":"*" }
        ],
      "permission": 31
  },
]
```

Details of the format for ACL are defined in Section 12. The ACL is composed of one or more ACEs. The ACL defines the access control policy for the Devices.

It should be noted that the ACL Resource requires the same security protection as other sensitive Resources, when it comes to both storage and handling by SRM and PSI. Thus hardening of an underlying Platform (HW and SW) must be considered for protection of ACLs and as explained below ACLs may have different scoping levels and thus hardening needs to be specially considered for each scoping level. For instance a physical device may host multiple Device implementations and thus secure storage, usage and isolation of ACLs for different Servers on the same Device needs to be considered.

5.1.1 ACL Architecture

The Server examines the Resource(s) requested by the client before processing the request. The access control resources (e.g. /oic/sec/acl, /oic/sec/acl2, etc…) are searched to find one or more ACE entries that match the requestor and the requested Resources. If a match is found then permission and period constraints are applied. If more than one match is found then the logical UNION of permissions is applied to the overlapping periods.

The server uses the connection context to determine whether the subject has authenticated or not and whether data confidentiality has been applied or not. Subject matching wildcard policies can match on each aspect. If the user has authenticated, then subject matching may happen at increased granularity based on role or device identity.

Each ACE contains the permission set that will be applied for a given Resource requestor. Permissions consist of a combination of CREATE, RETREIVE, UPDATE, DELETE and NOTIFY (CRUDN) actions. Requestors authenticate as a Device and optionally operating with one or more roles. Devices may acquire elevated access permissions when asserting a role. For example, an ADMINISTRATOR role might expose additional Resources and Interfaces not normally accessible.

5.1.1.1 Use of local ACLs

Servers may host ACL Resources locally. Local ACLs allow greater autonomy in access control processing than remote ACL processing by an Access Management Service (AMS) as described below.

The following use cases describe the operation of access control

Use Case 1: Server Device hosts 4 Resources (R1, R2, R3 and R4). Client Device D1 requests access to Resource R1 hosted at Server Device 5. ACL[0] corresponds to Resource R1 below and includes D1 as an authorized subject. Thus, Device D1 receives access to Resource R1 because the local ACL /oic/sec/acl/0 matches the request.

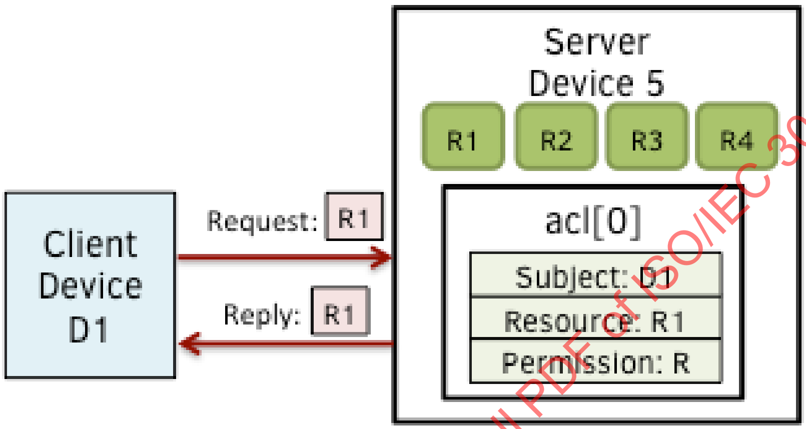


Figure 4 – Use case-1 showing simple ACL enforcement

Use Case 2: Client Device D2 access is denied because no local ACL match is found for subject D2 pertaining Resource R2 and no AMS policy is found.

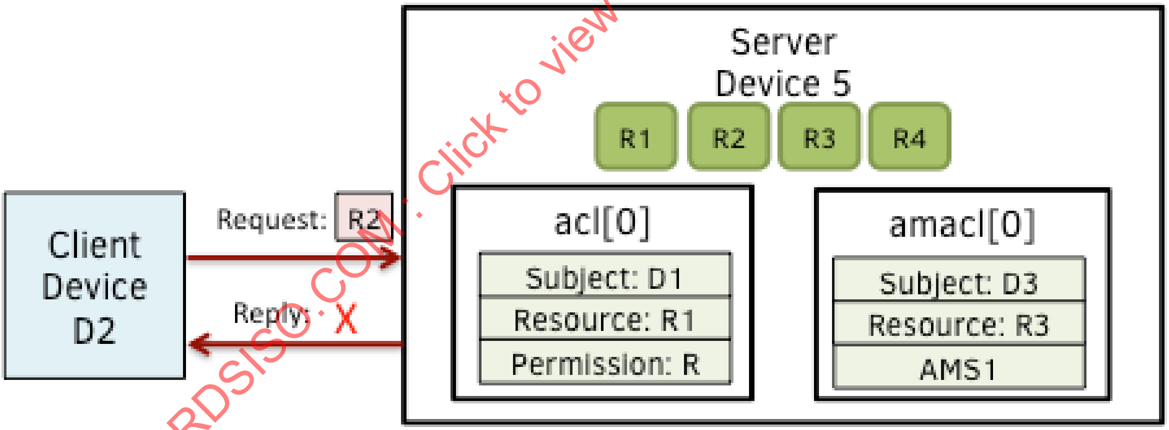


Figure 5 – Use case 2: A policy for the requested Resource is missing

5.1.1.2 Use of Access Manager Service

AMS improves ACL policy management. However, they can become a central point of failure. Due to network latency overhead, ACL processing may be slower through an AMS.

AMS centralizes access control decisions, but Server Devices retain enforcement duties. The Server shall determine which ACL mechanism to use for which Resource set. The /oic/sec/amacl Resource is an ACL structure that specifies which Resources will use an AMS to resolve access decisions. The /oic/sec/amacl may be used in concert with local ACLs (/oic/sec/acl).

The AMS is authenticated by referencing a credential issued to the device identifier contained in /oic/sec/acl2.rowneruuid.

The Server Device may proactively open a connection to the AMS using the Device ID found in /oic/sec/acl2.rowneruuid. Alternatively, the Server may reject the Resource access request with an error, ACCESS_DENIED_REQUIRES_SACL that instructs the requestor to obtain a suitable ACE policy using a SACL Resource /oic/sec/sacl. The /oic/sec/sacl signature may be validated using the credential Resource associated with the /oic/sec/acl2.rowneruuid.

The following use cases describe access control using the AMS:

Use Case 3: Device D3 requests and receives access to Resource R3 with permission Perm1 because the /oic/sec/amacl/0 matches a policy to consult the Access Manager Server AMS1 service

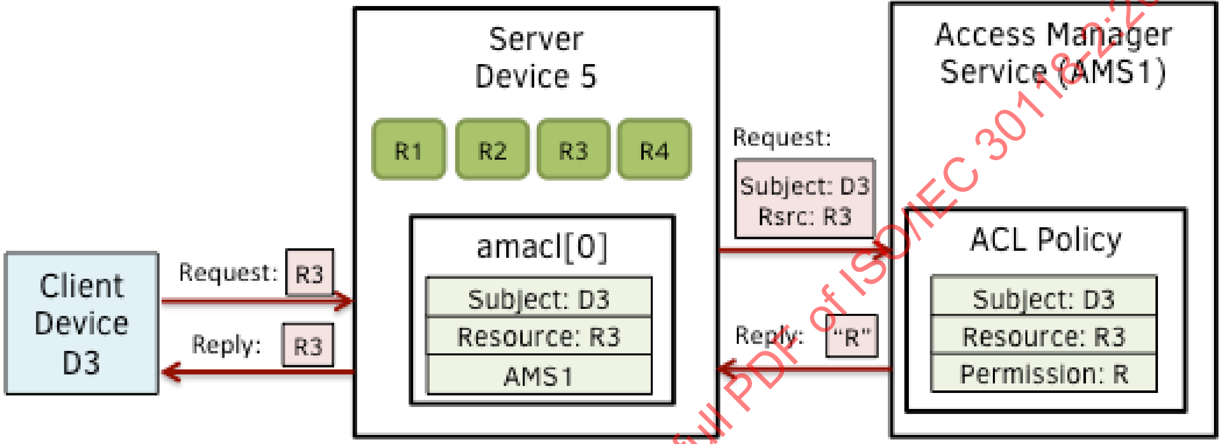


Figure 6 – Use case-3 showing Access Manager Service supported ACL

Use Case 4: Client Device D4 requests access to Resource R4 from Server Device 5, which fails to find a matching ACE and redirects the Client Device D4 to AMS1 by returning an error identifying AMS1 as a /oic/sec/sacl Resource issuer. Device D4 obtains Sacl1 signed by AMS1 and forwards the SACL to Server D5. D5 verifies the signature in the /oic/sec/sacl Resource and evaluates the ACE policy that grants Perm2 access.

ACE redirection may occur when D4 receives an error result with reason code indicating no match exists (i.e. ACCESS_DENIED_NO_ACE). D4 reads the /oic/sec/acl2 Resource to find the rowneruuid which identifies the AMS and then submits a request to be provisioned, in this example the AMS chooses to supply a SACL Resource, however it may choose to re-provision the local ACL Resources /oic/sec/acl and /oic/sec/acl2. The request is reissued subsequently. D4 is presumed to have been introduced to the AMS as part of Device onboarding or through subsequent credential provisioning actions.

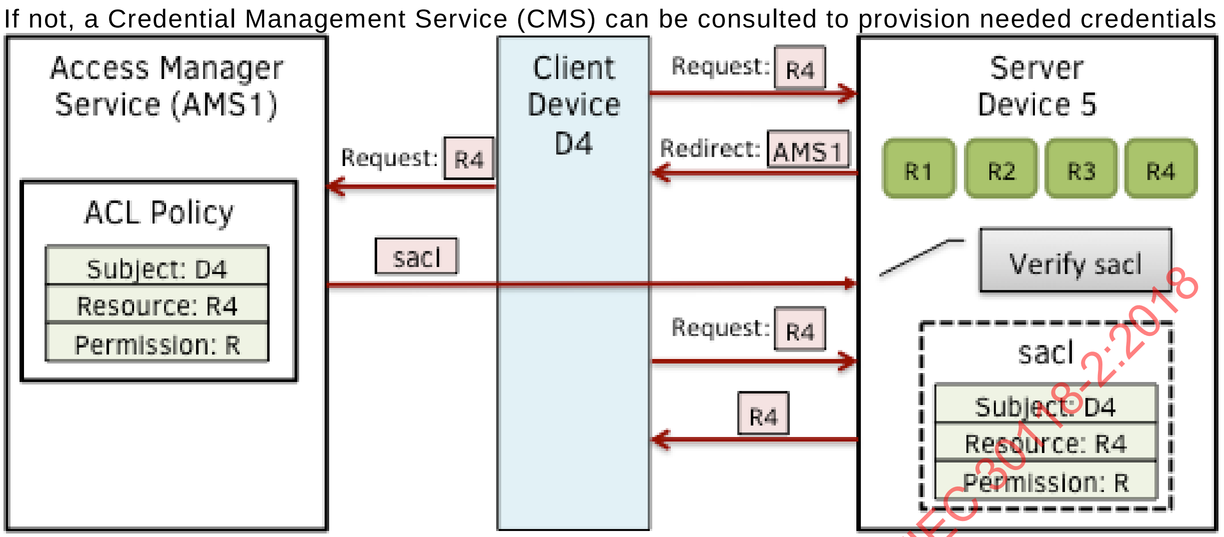


Figure 7 – Use case-4 showing dynamically obtained ACL from an AMS

5.1.2 Access Control Scoping Levels

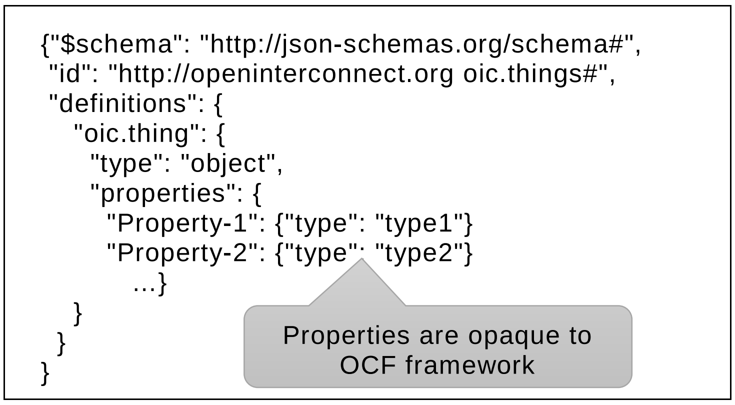
Group Level Access - Group scope means applying AC to the group of Devices that are grouped for a specific context. Group credentials may be used when encrypting data to the group or authenticating individual Device members into the group. Group Level Access means all group members have access to group data but non-group members must be granted explicit access. Group level access may also be specified using wildcard matching.

OCF Device Level Access – OCF Device scope means applying AC to an individual Device, which may contain multiple Resources. Device level access implies accessibility extends to all Resources available to the Device identified by DeviceID. Credentials used for AC mechanisms at Device are OCF Device-specific.

OCF Resource Level Access – OCF Resource level scope means applying AC to individual Resources. Resource access requires an ACL that specifies how the entity holding the Resource (Server) shall make a decision on allowing a requesting entity (Client) to access the Resource.

Property Level Access - Property level scope means applying AC only to a property that is part of a parent Resource. This is to provide a finer granularity for AC to Resources that may require different permissions for different properties. Property level access control is achieved by creating a Resource that contains a single property. This technique allows the Resource level access control mechanisms to be used to enforce access at a finer level of granularity than would otherwise be possible.

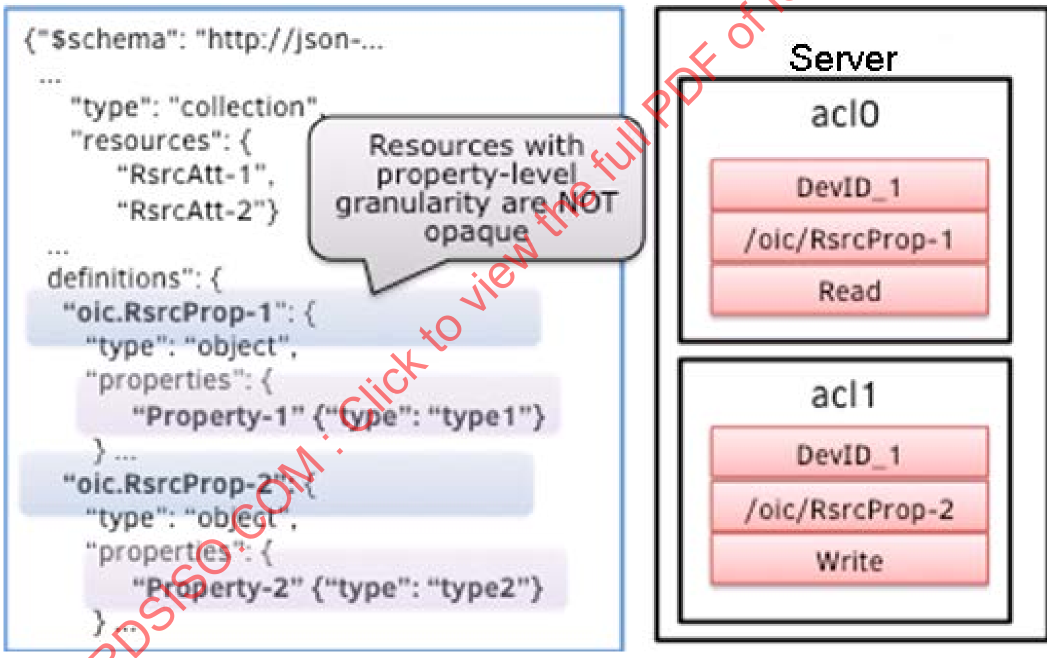
Controlling access to static Resources where it is impractical to redesign the Resource, it may appropriate to introduce a collection Resource that references the child Resources having separate access permissions. An example is shown below, where an "oic.thing" Resource has two properties: Property-1 and Property-2 that would require different permissions.



```
{"$schema": "http://json-schemas.org/schema#",
 "id": "http://openinterconnect.org oic.things#",
 "definitions": {
   "oic.thing": {
     "type": "object",
     "properties": {
       "Property-1": {"type": "type1"}
       "Property-2": {"type": "type2"}
          ...}
   }
  }
 }
```

Figure 8 – Example Resource definition with opaque Properties

Currently, OCF framework treats properly level information as opaque; therefore, different permissions cannot be assigned as part of an ACL policy (e.g. read-only permission to Property-1 and write-only permission to Property-2). Thus, the "oic.thing" is split into two new Resource "oic.RsrcProp-1" and "oic.RsrcProp-2". This way, property level ACL can be achieved through use of Resource-level ACLs.



```
{"$schema": "http://json-...
 ...
    "type": "collection",
    "resources": {
         "RsrcAtt-1",
         "RsrcAtt-2"}
  ...
 definitions": {
   "oic.RsrcProp-1": {
     "type": "object",
     "properties": {
         "Property-1" {"type": "type1"}
     } ...
   "oic.RsrcProp-2": {
     "type": "object",
     "properties": {
         "Property-2" {"type": "type2"}
     } ...
```

Figure 9 – Property Level Access Control

5.2 Onboarding Overview

Before a Device becomes operational in an OCF environment and is able to interact with other Devices, it needs to be appropriately onboarded. The first step in onboarding a Device is to configure the ownership where the legitimate user that owns/purchases the Device uses an Onboarding tool (OBT) and using the OBT uses one of the Owner Transfer Methods (OTM) to establish ownership. Once ownership is established, the OBT becomes the mechanism through which the Device can then be provisioned, at the end of which the Device becomes operational and is able to interact with other Devices in an OCF environment.

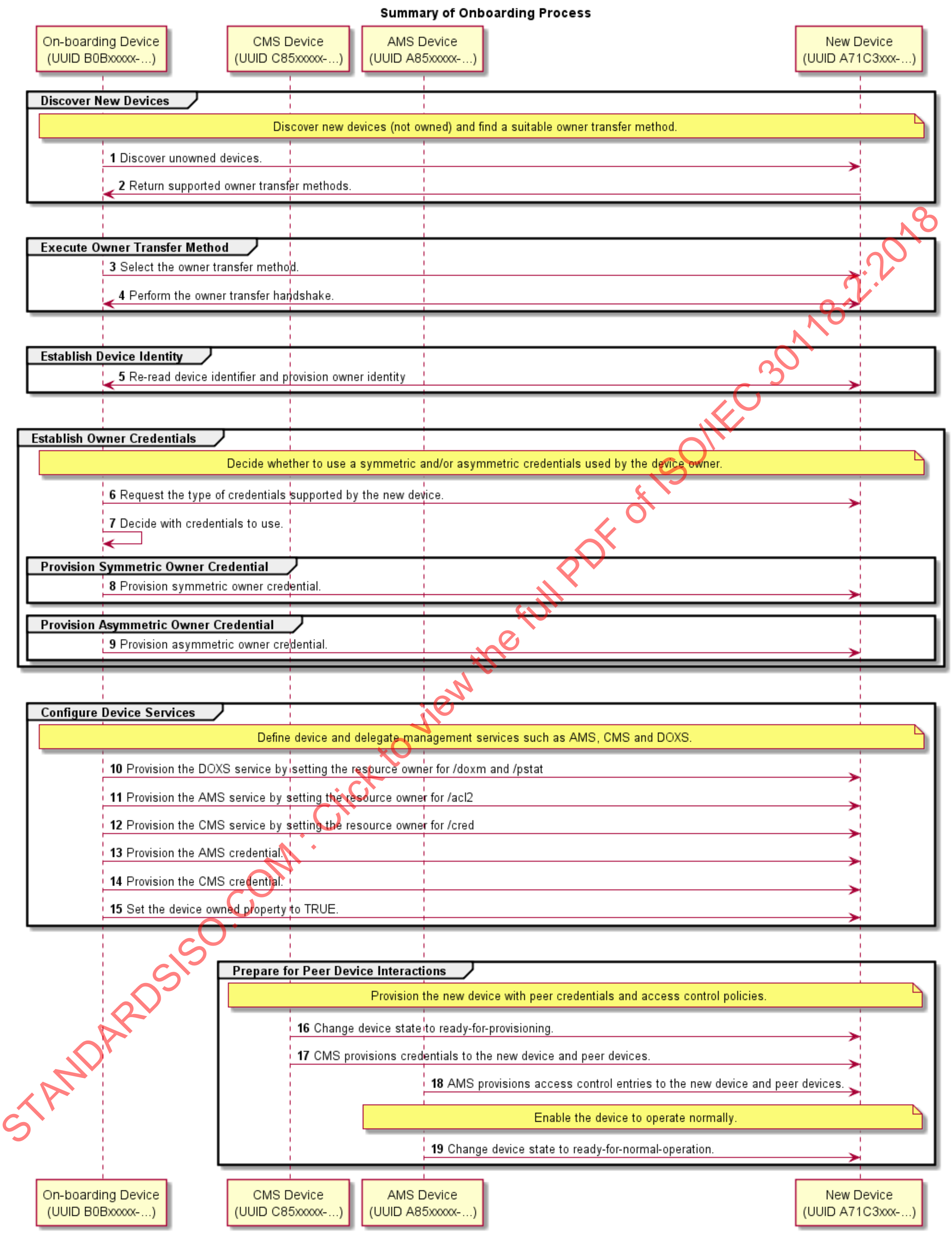

Summary of Onboarding Process
On-boarding Device (UUID B0Bxxxxx-...)
CMS Device (UUID C85xxxxx-...)
AMS Device (UUID A85xxxxx-...)
New Device (UUID A71C3xxx-...)
Discover New Devices
Discover new devices (not owned) and find a suitable owner transfer method.
1 Discover unowned devices.
2 Return supported owner transfer methods.
Execute Owner Transfer Method
3 Select the owner transfer method.
4 Perform the owner transfer handshake.
Establish Device Identity
5 Re-read device identifier and provision owner identity
Establish Owner Credentials
Decide whether to use a symmetric and/or asymmetric credentials used by the device owner.
6 Request the type of credentials supported by the new device.
7 Decide with credentials to use.
Provision Symmetric Owner Credential
8 Provision symmetric owner credential.
Provision Asymmetric Owner Credential
9 Provision asymmetric owner credential.
Configure Device Services
Define device and delegate management services such as AMS, CMS and DOXS.
10 Provision the DOXS service by setting the resource owner for /doxm and /pstat
11 Provision the AMS service by setting the resource owner for /acl2
12 Provision the CMS service by setting the resource owner for /cred
13 Provision the AMS credential.
14 Provision the CMS credential.
15 Set the device owned property to TRUE.
Prepare for Peer Device Interactions
Provision the new device with peer credentials and access control policies.
16 Change device state to ready-for-provisioning.
17 CMS provisions credentials to the new device and peer devices.
18 AMS provisions access control entries to the new device and peer devices.
Enable the device to operate normally.
19 Change device state to ready-for-normal-operation.

Figure 10 - Onboarding Overview

This section explains the onboarding and security provisioning process but leaves the provisioning of non-security aspects to other OCF specifications. In the context of security, all Devices are required to be provisioned with minimal security configuration that allows the Device to securely interact/communicate with other Devices in an OCF environment. This minimal security configuration is defined as the Onboarded Device "Ready for Normal Operation" and is specified in Section 8.

5.2.1 OnBoarding Steps

The flowchart below shows the typical steps that are involved during onboarding. Although onboarding may include a variety of non-security related steps, the diagram focus is mainly on the security related configuration to allow a new Device to function within an OCF environment. Onboarding typically begins with the Device getting "owned" by the legitimate user/system followed by configuring the Device for the environment that it will operate in. This would include setting information such as who can access the Device and what actions can be performed as well as what permissions the Device has for interacting with other Devices.

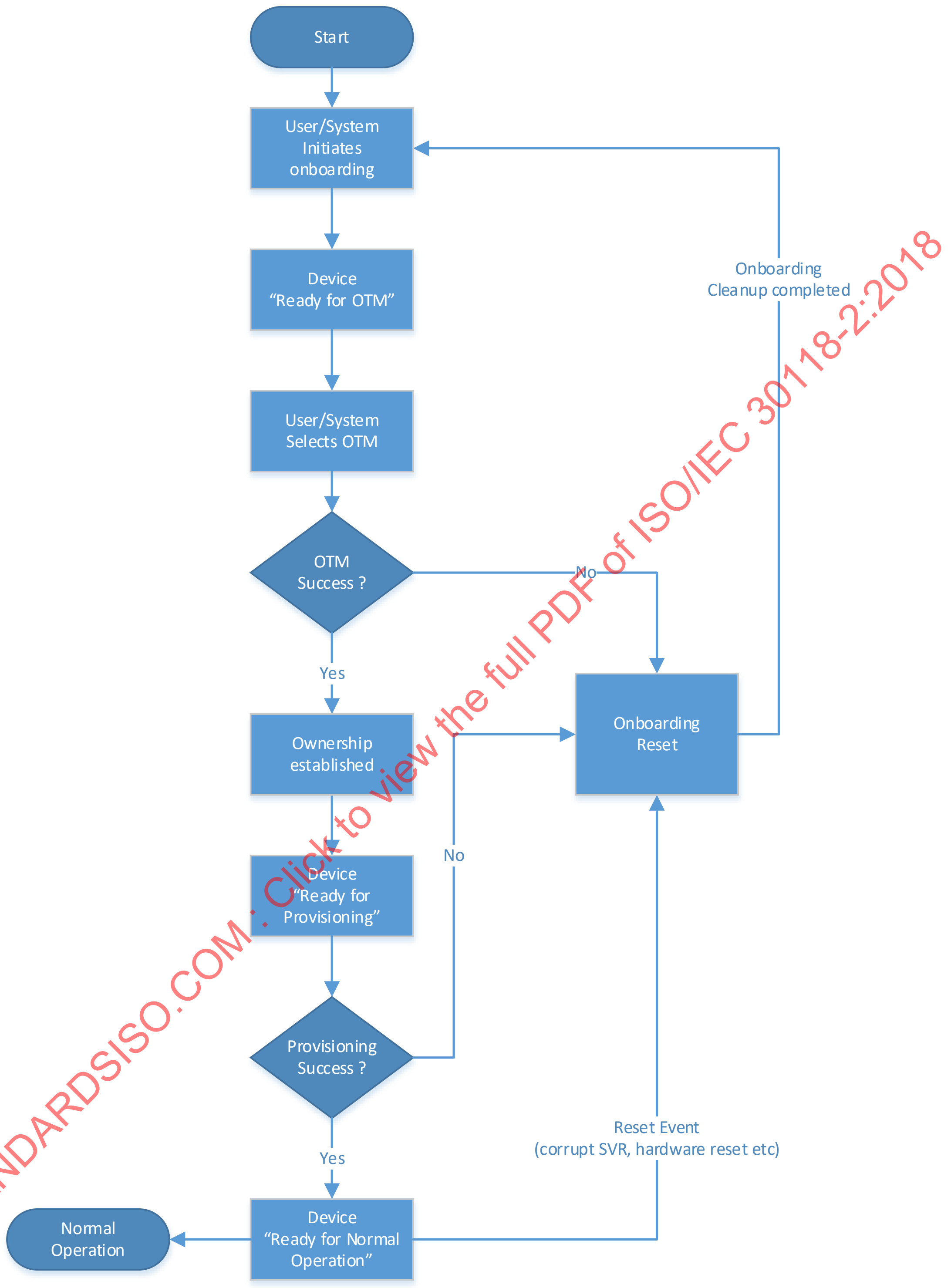

Start
User/System Initiates onboarding
Device "Ready for OTM"
User/System Selects OTM
OTM Success ?
No
Yes
Ownership established
Device "Ready for Provisioning"
Provisioning Success ?
No
Yes
Device "Ready for Normal Operation"
Normal Operation
Onboarding Reset
Onboarding Cleanup completed
Reset Event (corrupt SVR, hardware reset etc)

Figure 11 – OCF Onboarding Process

5.2.2 Establishing a Device Owner

The objective behind establishing Device ownership is to allow the legitimate user that owns/purchased the Device to assert itself as the owner and manager of the Device. This is done through the use of an OBT that includes the creation of an ownership context between the new Device and the OBT tool and asserts operational control and management of the Device. The OBT can be considered a logical entity hosted by tools/ Servers such as a network management console, a device management tool, a network-authoring tool, a network provisioning tool, a home gateway device, or a home automation controller. A physical device hosting the OBT will be subject to some security hardening requirements, thus preserving integrity and confidentiality of any credentials being stored. The tool/Server that establishes Device ownership is referred to as the OBT.

The OBT uses one of the Owner Transfer method specified in Section 7.3 to securely establish Device ownership. The term owner transfer is used since it is assumed that even for a new Device, the ownership is transferred from the manufacturer/provider of the Device to the buyer/legitimate user of the new Device.

An owner transfer method establishes a new owner (the operator of OBT) that is authorized to manage the Device. Owner transfer establishes the following

- An Owner Credential (OC) that is provisioned by the OBT in the /oic/sec/doxm Resource of the Device. This OC allows the Device and OBT to mutually authenticate during subsequent interactions. The OC asserts the user/system's ownership of the Device by recording the credential of the OBT as the owner. The OBT also records the identity of Device as part of ownership transfer.
- The Device owner establishes trust in the Device through the OTM.
- Preparing the Device for provisioning by providing credentials that may be needed.

5.2.2.1 Preparing the Device for provisioning

Once Device ownership is established, the Device needs to be prepared for provisioning. This could be in the form of getting bootstrapping parameters (BP) that allow the Device to contact the bootstrap server (BS) and establish a secure session with the BS. The typical bootstrap parameters are as follows

- Bootstrap server (BS)/ tool metadata: This information needs to include addressing and access mechanism/ protocol to be used to access the bootstrap server. Addressing information may include Server URI or FQDN if HTTP or TCP/IP is being used to contact the Server.
- Bootstrapping credential (BC): This is the credential that the Device needs to use to contact the BS, authenticate to the BS, and establish a secure session with the BS to receive provisioning parameters from the BS. The BC may be derived from the OC depending on the type of OC.

If the OBT itself acts as the bootstrap server, the metadata for the bootstrap server may point to a service hosted by the OBT itself. In such a scenario additional BC credentials may not be needed.

5.2.3 Provisioning for Normal Operation

Once the Device has the necessary information to initiate provisioning, the next step is to provision additional security configuration that allows the Device to become operational. This can include setting various parameters and may also involve multiple steps. For example if the Device is to receive its configuration from a bootstrapping server, then the provisioning may involve connecting to the bootstrap server and receive its configuration. Also provisioning of ACL's for the various Resources hosted by the Server on the Device is done at this time. Note that the provisioning step is not limited to this stage only. Device provisioning can happen at multiple stages in the Device's operational lifecycle. However specific security related provisioning of Resource and Property state would likely happen at this stage at the end of which, each Device reaches the Onboarded Device "Ready for Normal Operation" State. The "Ready for Normal Operation" State is expected to be consistent and well defined regardless of the specific OTM used or regardless of the variability in what gets provisioned. However individual OTM mechanisms and provisioning steps may specify additional configuration of Resources and Property states. The minimal mandatory configuration required for a Device to be in "Ready for Normal Operation" state is specified in Section 8.

5.3 Provisioning

Note that in general, provisioning may include processes during manufacturing and distribution of the Device as well as processes after the Device has been brought into its intended environment (parts of onboarding process). In this specification, security provisioning includes, processes after ownership transfer (even though some activities during ownership transfer and onboarding may lead to provisioning of some data in the Device) configuration of credentials for interacting with bootstrapping and provisioning services, configuration of any security related Resources and credentials for dealing with any services that the Device need to contact later on.

Once the ownership transfer is complete and bootstrap credentials are established, the Device needs to engage with the bootstrap server to be provisioned with proper security credentials and parameters. These parameters can include

- Security credentials through a credential management service (CMS), currently assumed to be deployed in the same OBT.
- Access control policies and ACLs through an ACL provisioning service (APS), currently assumed to be deployed in the same OBT, but may be part of AMS in future.

As mentioned, to accommodate a scalable and modular design, these functions are considered as services that in future could be deployed as separate servers. Currently, the deployment assumes that these services are all deployed as part of a OBT. Regardless of physical deployment scenario, the same security-hardening requirement) applies to any physical server that hosts the tools and security provisioning services discussed here.

Devices are *aware* of their security provisioning status. Self-awareness allows them to be proactive about provisioning or re-provisioning security Resources as needed to achieve the devices operational goals.

5.3.1 Provisioning a bootstrap service

The Device is discovered or programmed with the bootstrap parameters (BP), including the metadata required to discover and interact with the Bootstrap server (BS). The Device is configured with bootstrap credential (BC) required to communicate with BS securely.

In the Resource structure, the rowneruuid Property in the /oic/sec/pstat Resource identifies the bootstrap service (BSS).

When symmetric keys are used, the OC is used to derive the BC. However, when the Device is capable of using asymmetric keys for ownership transfer and other provisioning processes, there may not be a need for a cryptographic relationship between BC and OC.

Regardless of how the BC is created, the communication between Device and BS (and potentially other servers) must be done securely. For instance when a pre-shared key is used for secure connection with the Device, the oic.sec.bss service includes the oic.sec.cred Resource is provisioned with the PSK.

5.3.2 Provisioning other services

To be able to support the use of potentially different device management service hosts, each Device Secure Virtual Resource (SVR) has an associated Resource owner. The onboarding Device, also known as DOXS, provisions rowneruuid Properties with the appropriate provider identity.

- Credential Management Service (CMS) : (/oic/sec/cred.rowneruuid)
- Access Manager Service (AMS) : (/oic/sec/acl.rowneruuid and /oic/sec/acl2.rowneruuid)

When these services are populated the Device may proactively request provisioning and verify provisioning requests are authorized. Each of the services above must be performed securely and thus require specific credentials to be provisioned. The DOXS service may initiate of any services above by signaling the service provider Device(s) or by setting the appropriate vector in the 'tm' Property of the Device's /oic/sec/pstat Resource. This will cause the Device to re-provision its credential and or access Resources

5.3.3 Credential provisioning

Several types of credential may be configured in a /oic/sec/cred Resource. Currently, they include at least the following credential types; pairwise symmetric keys, group symmetric keys, certificates, asymmetric keys and signed asymmetric keys. Keys may be provisioned by a CMS (e.g. "oic.sec.cms") or dynamically using a Diffie-Hellman key agreement protocol or through other means.

The following describe an example on how a Device can update a PSK for a secure connection. A Device may discover the need to update credentials, e.g. because a secure connection attempt fails. The Device will then need to request credential update from a CMS. The Device may enter credential-provisioning mode (e.g. /oic/sec/pstat.Cm=16) and may configure operational mode (e.g. /oic/sec/pstat.Om="1") to request an update to its credential Resource. The CMS responds with a new pairwise pre-shared key (PSK).

5.3.4 Role assignment and provisioning

The Servers, receiving requests for Resources they host, need to examine the role asserted by the Client requesting the Resource and compare that role with the constraints described in their ACLs corresponding to the services. Thus, a Client Device may need to be provisioned with one or more role credentials.

Each Device holds the role information as a Property within the credential Resource. Thus, it is possible that a Client, seeking a role provisioning, enters a mode where both its credentials and ACLs can be provisioned (if they are provisioned by the same sever!). The provisioning mode/status is typically indicated by the content of /oic/sec/pstat.

Once provisioned, the Client can assert the role it is using as described in Section 10.3.1, if it has a certificate role credential.

Alternatively, if the server has been provisioned with role information for a client, or the client has previously asserted roles to the server, the client can assert a specific role with the CoAP payload:

e.g. GET /a/light?roleid={“role”:”Role-A”}

The client has no way to know in advance what roles are provisioned on the server, and must attempt an action and observe the server's response. If the response is permission denied, the client learns that either the server is not provisioned with the role, or the ACLs are misconfigured. If no specific role is specified in the CoAP payload, all provisioned roles are used in ACL enforcement. When a server has multiple roles provisioned for a client, access to a Resource is granted if it would be granted under any of the roles.

5.3.5 ACL provisioning

During ACL provisioning, the Device establishes a secure connection to an APS (or bootstrap server, if it is hosting the APS). The APS will instantiate or update Device ACLs according to the ACL policy.

The Device and APS may establish an observer relationship such that when a change to the ACL policy is detected; the Device is notified triggering ACL provisioning.

The APS (e.g. rt="oic.sec.aps") may digitally sign an ACL as part of issuing a /oic/sec/sacl Resource. The public key used by Servers to verify the signature may be provisioned as part of credential provisioning. A /oic/sec/cred Resource with an asymmetric key type or signed asymmetric key type is used. The PublicData Property contains the APS's public key.

5.4 Secure Resource Manager-(SRM)

SRM plays a key role in the overall security operation. In short, SRM performs both management of SVR and access control for requests to access and manipulate Resources. SRM consists of 3 main functional elements:

- A Resource manager (RM): responsible for 1) Loading SVRs from persistent storage (using PSI) as needed. 2) Supplying the Policy Engine (PE) with Resources upon request. 3) Responding to requests for SVRs. While the SVRs are in SRM memory, the SVRs are in a format that is consistent with device-specific data store format. However, the RM will use JSON format to marshal SVR data structures before be passed to PSI for storage, or travel off-device.

- A Policy Engine (PE) that takes requests for access to SVRs and based on access control policies responds to the requests with either "ACCESS_GRANTED" or "ACCESS_DENIED". To make the access decisions, the PE consults the appropriate ACL and looks for best Access Control Entry (ACE) that can serve the request given the subject (Device or role) that was authenticated by DTLS.

- Persistent Storage Interface (PSI): PSI provides a set of APIs for the RM to manipulate files in its own memory and storage. The SRM design is modular such that it may be implemented in the Platform's secure execution environment; if available.

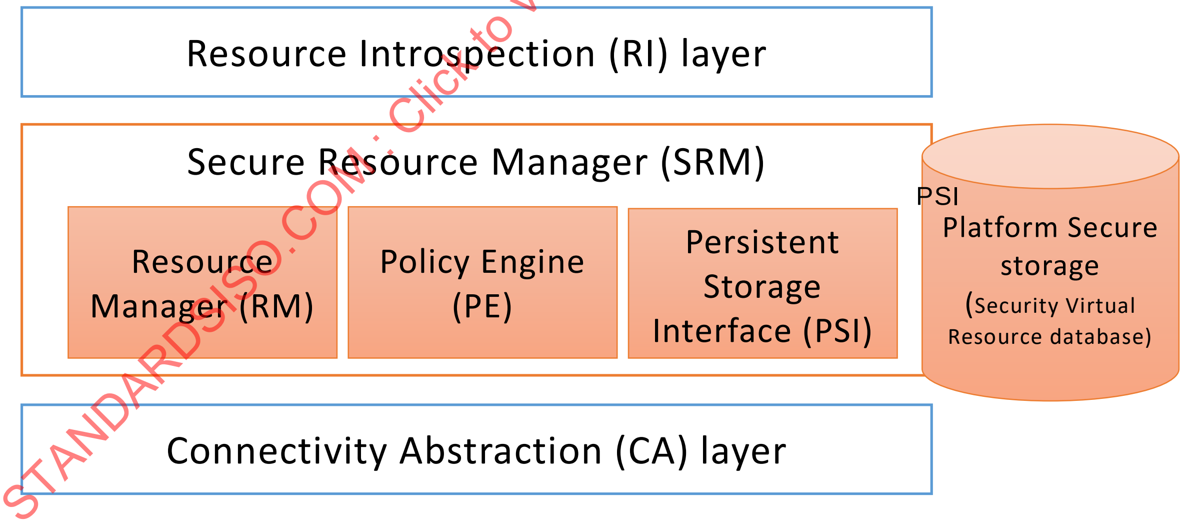

Resource Introspection (RI) layer
Secure Resource Manager (SRM)
Resource Manager (RM)
Policy Engine (PE)
Persistent Storage Interface (PSI)
PSI
Platform Secure storage
(Security Virtual Resource database)
Connectivity Abstraction (CA) layer

Figure 12 – OCF's SRM Architecture

5.5 Credential Overview

Devices may use credentials to prove the identity and role(s) of the parties in bidirectional communication. Credentials can be symmetric or asymmetric. Each device stores secret and public parts of its own credentials where applicable, as well as credentials for other devices that have been provided by the OBT or a CMS. These credentials are then used in the establishment of secure communication sessions (e.g. using DTLS) to validate the identities of the participating parties. Role credentials are used once an authenticated session is established, to assert one or more roles for a device.

6 Security for the Discovery Process

The main function of a discovery mechanism is to provide Universal Resource Identifiers (URIs, called links) for the Resources hosted by the Server, complemented by attributes about those Resources and possible further link relations. (in accordance to Section 10 in OCF Core Specification)

6.1 Security Considerations for Discovery

When defining discovery process, care must be taken that only a minimum set of Resources are exposed to the discovering entity without violating security of sensitive information or privacy requirements of the application at hand. This includes both data included in the Resources, as well as the corresponding metadata.

To achieve extensibility and scalability, this specification does not provide a mandate on discoverability of each individual Resource. Instead, the Server holding the Resource will rely on ACLs for each Resource to determine if the requester (the Client) is authorized to see/handle any of the Resources.

The /oic/sec/acl2 Resource contains ACL entries governing access to the Server hosted Resources. (See Section 13.4)

Aside from the privacy and discoverability of Resources from ACL point of view, the discovery process itself needs to be secured. This specification sets the following requirements for the discovery process:

1. Providing integrity protection for discovered Resources.

2. Providing confidentiality protection for discovered Resources that are considered sensitive.

The discovery of Resources is done by doing a RETRIEVE operation (either unicast or multicast) on the known Resource "/oic/res".

The discovery request is sent over a non-secure channel (multicast or unicast without DTLS), a Server cannot determine the identity of the requester. In such cases, a Server that wants to authenticate the Client before responding can list the secure discovery URI (e.g. coaps://IP:PORT/oic/res) in the unsecured /oic/res response. This means the secure discovery URI is by default discoverable by any Client. The Client will then be required to send a separate unicast request using DTLS to the secure discovery URI.

For secure discovery, any Resource that has an associated ACL2 will be listed in the response to /oic/res if and only if the Client has permissions to perform at least one of the CRUDN operations (i.e. the bitwise OR of the CRUDN flags must be true).

For example, a Client with DeviceId "d1" makes a RETRIEVE request on the "/door" Resource hosted on a Server with DeviceId "d3" where d3 has the ACL2s below:

```
{
    "aclist2": [
      {
        "subject": {"uuid": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"},
        "resources": [{"href":"/door"}],
        "permission": 2, // RETRIEVE
        "aceid": 1
      }
    ],
    "rowneruuid": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"
}
```

```
{
    "aclist2": [
      {
        "subject": {"authority": "owner", "role": "owner"}
        "resources": [{"href":"/door"}],
        "permission": 2, // RETRIEVE
        "aceid": 2
      }
    ],
    "rowneruuid": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"
}
{
    "aclist2": [
      {
        "subject": {"uuid": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"},
        "resources": [{"href":"/door/lock"}],
        "permission": 4, // UPDATE
        "aceid": 3
      }
    ],
    "rowneruuid": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"
}
{
    "aclist2": [
      {
        "subject": {"conntype": "anon-clear"},
        "resources": [{"href":"/light"}],
        "permission": 2, // RETRIEVE
        "aceid": 4
      }
    ],
    "rowneruuid": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"
}
```

The ACL indicates that Client "d1" has RETRIEVE permissions on the Resource. Hence when device "d1" does a discovery on the /oic/res Resource of the Server "d3", the response will include the URI of the "/door" Resource metadata. Client "d2" will have access to both the Resources. ACE2 will prevent "d4" from update.

Discovery results delivered to d1 regarding d3's /oic/res Resource from the secure Interface:

```
[
  {
    "href": "/door",
    "rt": ["oic.r.door"],
    "if": ["oic.if.b", "oic.ll"],
     "di": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1",
  }
]
```

Discovery results delivered to d2 regarding d3's /oic/res Resource from the secure Interface:

```
[
  {
    "href": "/door",
    "rt": ["oic.r.door"],
    "if": ["oic.if.b", "oic.ll"],
     "di": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"
  },
  {
    "href": "/door/lock",
    "rt": ["oic.r.lock"],
    "if": ["oic.if.b"],
    "type": ["application/json", "application/exi+xml"]
  }
]
```

Discovery results delivered to d4 regarding d3’s /oic/res Resource from the secure Interface:

```
[
 {
    "href": "/door/lock",
    "rt": ["oic.r.lock"],
    "if": ["oic.if.b"],
    "type": ["application/json", "application/exi+xml"],
    "di": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"
  }
]
```

Discovery results delivered to any device regarding d3’s /oic/res Resource from the unsecure Interface:

```
[
 {
    "di": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1",
    "href": "/light",
    "rt": ["oic.r.light"],
    "if": ["oic.if.s"]
 }
]
```

7 Security Provisioning

7.1 Device Identity

Each Device, which is a logical device, is identified with a device ID.

Devices shall be identified by a Device ID value that is established as part of device onboarding. The /oic/sec/doxm Resource specifies the Device ID format (e.g. urn:uuid). Device IDs shall be unique within the scope of operation of the corresponding OCF network, and should be universally unique. Device ID uniqueness within the network shall be enforced at device onboarding. A Device OBT shall verify the chosen new device identifier does not conflict with other devices previously introduced into the network.

Devices maintain an association of Device ID and cryptographic credential using a /oic/sec/cred Resource. Devices regard the /oic/sec/cred Resource as authoritative when verifying authentication credentials of a peer device.

A Device maintains its device ID in the /oic/sec/doxm Resource. It maintains a list of credentials, both its own and other device credentials, in the /oic/sec/cred Resource. The device ID can be used to distinguish between a device's own credential, and credentials for other devices. Furthermore, the /oic/sec/cred Resource may contain multiple credentials for the device.

Device ID shall be:

- Unique
- Immutable
- Verifiable

When using manufacturer certificates, the certificate should bind the ID to the stored secret in the device as described later in this section.

A physical device, referred to as a Platform in OCF specifications, may host multiple Devices. The Platform is identified by a Platform ID. The Platform ID shall be globally unique and inserted in the device in an integrity protected manner (e.g. inside secure storage or signed and verified).

Note: An OCF Platform may have a secure execution environment, which shall be used to secure unique identifiers and secrets. If a Platform hosts multiple devices, some mechanism is needed to provide each Device with the appropriate and separate security.

7.1.1 Device Identity for Devices with UAID

When a manufacturer certificate is used with certificates chaining to an OCF root CA (as specified in Section 7.1.1), the manufacturer shall include a Platform ID inside the certificate subject CN field. In such cases, the device ID may be created according to the Unique Authenticable IDentifier (UAID) scheme defined in this section.

For identifying and protecting Devices, the Platform Secure Execution Environment (SEE) may opt to generate new Dynamic Public Key Pair (DPKP) for each Device it is hosting, or it may opt to simply use the same public key credentials embedded by manufacturer; Embedded Platform Credential (EPC). In either case, the Platform SEE will use its Random Number Generator (RNG) to create a device identity called UAID for each Device. The UAID is generated using eitherEPC only or the combnation of DPC and EPC if both are available. When both are available, the Platform shall use both key pairs to generate the UAID as described in this section.

The Device ID is formed from the device's public keys and associated OCF Cipher Suite. The Device ID is formed by:

1. Determining the OCF Cipher Suite of the Dynamic Public Key. The Cipher Suite curve must match the usage of the AlgorithmIdentifier used in SubjectPublicKeyInfo as intended for use with Device security mechanisms. Use the encoding of the CipherSuite as the 'csid' value in the following calculations. Note that if the OCF Cipher Suite for Dynamic Public key is different from the ciphersuite indicated in the Platform certificate (EPC), the OCF Cipher Suite shall be used below.

2. From EPC extract the value of embedded public key. The value should correspond to the value of subjectPublicKey defined in SubjectPublicKeyInfo of the certificate. In the following we refer to this as EPK. If the public key is extracted from a certificate, validate that the AlgorithmIdentifier matches the expected value for the CipherSuite within the certificate.

3. From DPC Extract the value of the public key. The value should correspond to the value of subjectPublicKey defined in SubjectPublicKeyInfo. In the following we refer to this as DPK.

4. Using the hash for the Cipher Suite calculate:
   h = hash('uaid' | csid | EPK| DPK | <other_info>)

Other_info could be 1) device type as indicated in /oic/d (could be read-only and set by manufacturer), 2) in case there are two sets of public key pairs (one embedded, and one dynamically generated), both public keys would be included.

5. Truncate to 128 bits by taking the first 128 bits of h
   UAID = h[0:16] # leftmost 16 octets

6. Convert the binary UAID to a ASCII string by
   USID = base27encode(UAID)

```
def base_N_encode(octets, alphabet):
long_int = string_to_int( octets )
    text_out = ''
    while long_int > 0:
        long_int, remainder = divmod(long_int, len(alphabet))
        text_out = alphabet[remainder] + text_out
    return text_out

b27chars = 'ABCDEFGHJKMNPQRTWXYZ2346789'
def b27encode(octet_string):
    """Encode a octet string using 27 characters. """

    return base_N_encode(octet_string, _b27chars )
```

7. Append the string value of USID to 'urn:usid:' to form the final string value of the Device ID
   urn:usid:ABXW....

Whenever the public key is encoded the format described in RFC 7250 for SubjectPublicKeyInfo shall be used.

7.1.1.1 Validation of UAID

To be able to use the newly generated Device ID (UAID) and public key pair (DPC), the device Platform shall use the embedded private key (corresponding to manufacturer embedded public key and certificate) to sign a token vouching for the fact that it (the Platform) has in fact generated the DPC and UAID and thus deferring the liability of the use of the DPC to the new device owner. This also allows the ecosystem to extend the trust from manufacturer certificate to a device issued certificate for use in the new DPC and UAID. The degree of trust is in dependent of the level of hardening of the device SEE.

Dev_Token=Info, Signature(hash(info))

Signature algorithm=ECDSA (can be same algorithm as that in EPC or that possible for DPC)

Hash algorithm=SHA256

Info=UAID| <Platform ID> | UAID_generation_data | validity

UAID_generation_data=data passed to the hash algorithm used to generate UAID.

Validity=validity period in days (how long the token will be valid)

7.2 Device Ownership

This is an informative section. Devices are logical entities that are security endpoints that have an identity that is authenticable using cryptographic credentials. A Device is ‘un-owned’ when it is first initialized. Establishing device ownership is a process by which the device asserts it’s identity to an OBT and the OBT asserts its identity to the device. This exchange results in the device changing its ownership state, thereby preventing a different OBT from asserting administrative control over the device.

The ownership transfer process starts with the OBT discovering a new device that is "un-owned" through examination of the "Owned" Property of the /oic/sec/doxm Resource of the new device. At the end of ownership transfer, the following is accomplished:

1. Establish a secure session between new device and the OBT.
2. Optionally asserts any of the following:
    a. Proximity (using PIN) of the OBT to the Platform.
    b. Manufacturer’s certificate asserting Platform vendor, model and other Platform specific attributes.
3. Determines the device identifier.
4. Determines the device owner.
5. Specifies the device owner (e.g. Device ID of the OBT).
6. Provisions the device with owner’s credentials.
7. Sets the ‘Owned’ state of the new device to TRUE.

7.3 Device Ownership Transfer Methods

7.3.1 OTM implementation requirements

This document provides specifications for several methods for ownership transfer. Implementation of each individual ownership transfer method is considered optional. However, each device shall implement at least one of the ownership transfer methods not including vendor specific methods.

All owner transfer methods (OTMs) included in this document are considered optional. Each vendor is required to choose and implement at least one of the OTMs specified in this specification. The OCF, does however, anticipate vendor-specific approaches will exist. Should the vendor wish to have interoperability between an vendor-specific owner transfer method and and OBTs from other vendors, the vendor must work directly with OBT vendors to ensure interoperability. Notwithstanding, standardization of OTMs is the preferred approach. In such cases, a set of guidelines is provided below to help vendors in designing vendor-specific OTMs. (See Section 7.3.6).

The device owner transfer method (doxm) Resource is extensible to accommodate vendor-defined methods. All OTM methods shall facilitate allowing the OBT to determine which OC is most appropriate for a given new device within the constraints of the capabilities of the device. The OBT will query the credential types that the new device supports and allow the OBT to select the credential type from within device constraints.

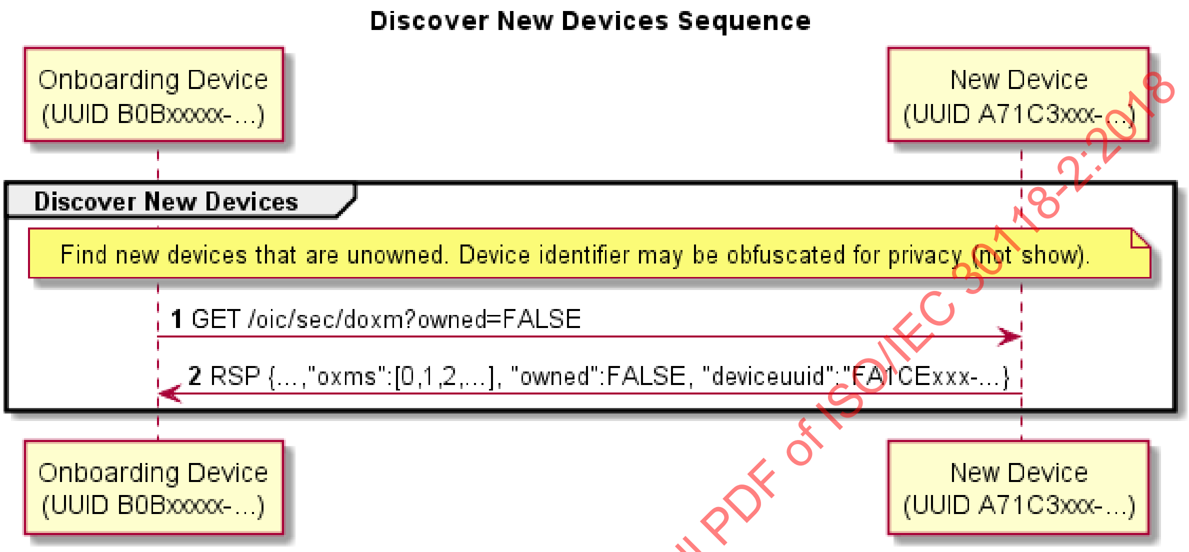

Discover New Devices Sequence
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Discover New Devices
Find new devices that are unowned. Device identifier may be obfuscated for privacy (not show).
1 GET /oic/sec/doxm?owned=FALSE
2 RSP {...,"oxms":[0,1,2,...], "owned":FALSE, "deviceuuid":"FA1CExxx-...}
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

Figure 13 - Discover New Device Sequence

Step	Description
1	The OBT queries to see if the new device is not yet owned.
2	The new device returns the /oic/sec/doxm Resource containing ownership status and supported owner transfer methods. It also contains a temporal device ID that may change subsequent to successful owner transfer. The device should supply a temporal ID to facilitate discovery as a guest device. Section 7.3.9 provides security considerations regarding selecting an owner transfer method.

Table 2 - Discover New Device Details

Vendor-specific device owner transfer methods shall adhere to the /oic/sec/doxm Resource specification for OCs that results from vendor-specific device owner transfer. Vendor-specific methods should include provisions for establishing trust in the new device by the OBT an optionally establishing trust in the OBT by the new device.

The end state of a vendor-specific owner transfer method shall allow the new device to authenticate to the OBT and the OBT to authenticate to the new device.

Additional provisioning steps may be applied subsequent to owner transfer success leveraging the established session, but such provisioning steps are technically considered provisioning steps that an OBT may not anticipate hence may be invalidated by OBT provisioning.

7.3.2 SharedKey Credential Calculation

The SharedKey credential is derived using a PRF that accepts the key_block value resulting from the DTLS handshake used for onboarding. The Server and Device OBT shall use the following calculation to ensure interoperability across vendor products:

SharedKey = *PRF*(Secret, Message);

Where:

- PRF shall use TLS 1.2 PRF defined by RFC5246 section 5.
- Secret is the key_block resulting from the DTLS handshake
  - See RFC5246 Section 6.3
  - The length of key_block depends on cipher suite.
    - (e.g. 96 bytes for TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
      40 bytes for TLS_PSK_WITH_AES_128_CCM_8)
- Message is a concatenation of the following:
  - DoxmType string for the current onboarding method (e.g. "oic.sec.doxm.jw")
    - See "Section 13.1.1 OCF defined owner transfer methods for specific DoxmTypes"
  - OwnerID is a UUID identifying the device owner identifier and the device that maintains SharedKey.
    - Use raw bytes as specified in RFC4122 section 4.1.2
  - Device ID is new device's UUID Device ID
    - Use raw bytes as specified in RFC4122 section 4.1.2
- SharedKey Length will be 32 octets.
  - If subsequent DTLS sessions use 128 bit encryption cipher suites the leftmsot 16 octets will be used. DTLS sessions using 256 bit encryption cipher suites will use all 32 octets.

7.3.3 Certificate Credential Generation

The Certificate Credential will be used by Devices for secure bidirectional communication. The certificates will be issued by a CMS or an external certificate authority (CA). This CA will be used to mutually establish the authenticity of the Device. The onboarding details for certificate generation will be specified in a later version of this specification.

7.3.4 Just-Works Owner Transfer Method

Just-works owner transfer method creates a symmetric key credential that is a pre-shared key used to establish a secure connection through which a device should be provisioned for use within the owner's network. Provisioning additional credentials and Resources is a typical step following ownership establishment. The pre-shared key is called SharedKey.

The ownership transfer process starts with the OBT discovering a new device that is "un-owned" through examination of the "owned" Property of the /oic/sec/doxm Resource at the Device hosted by the new device.

Once the OBT asserts that the device is un-owned, when performing the Just-works owner transfer method, the OBT relies on DTLS key exchange process where an anonymous Elliptic Curve Diffie-Hellman (ECDH) is used as a key agreement protocol.

The following OCF-defined vendor-specific ciphersuites are used for the Just-works owner transfer method.

TLS_ECDH_ANON_WITH_AES_128_CBC_SHA256,
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA256

These are not registered in IANA, the ciphersuite values are assigned from the reserved area for private use (0xFF00 ~ 0xFFFF). The assigned values are 0xFF00 and 0xFF01, respectively.

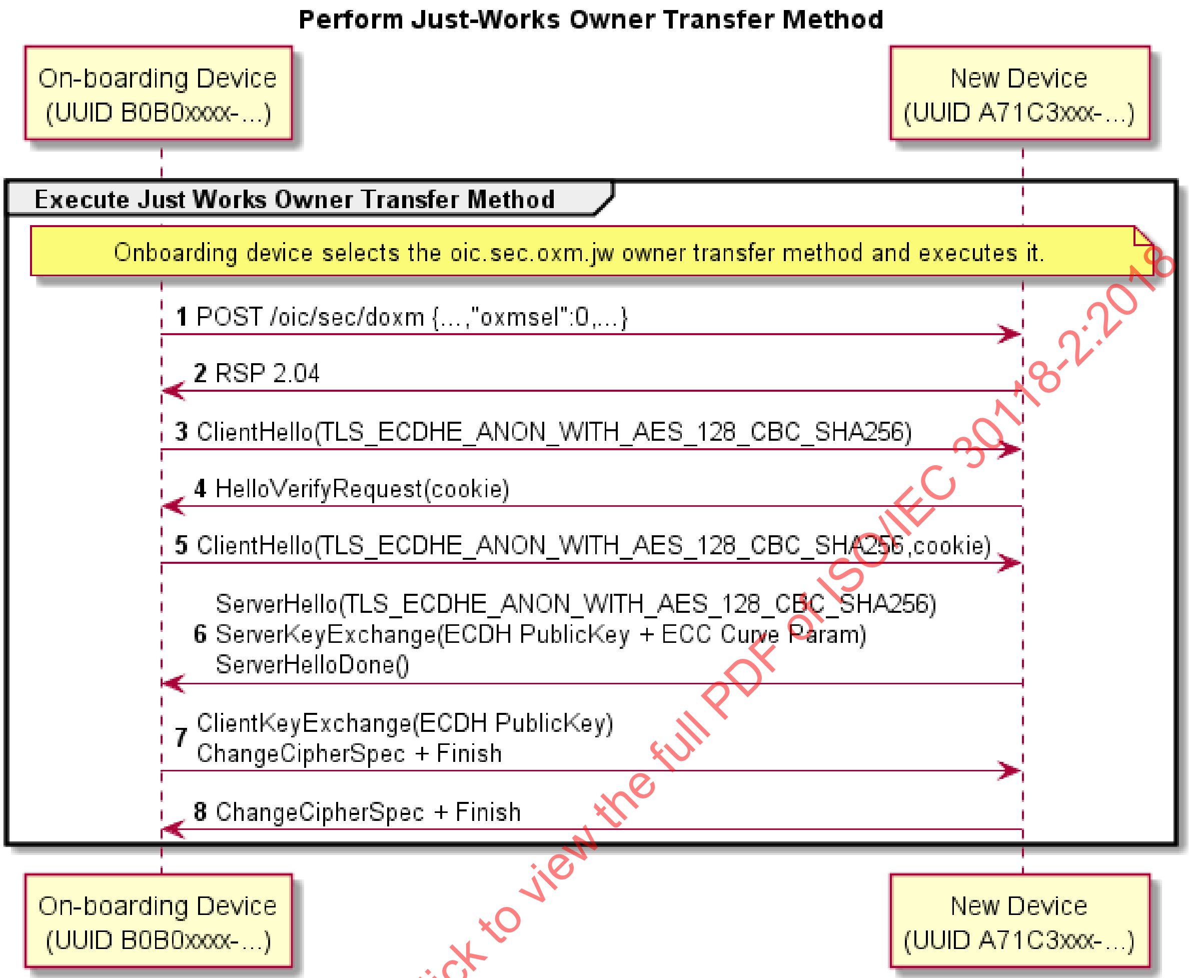

Perform Just-Works Owner Transfer Method
On-boarding Device (UUID B0B0xxxx-...)
New Device (UUID A71C3xxx-...)
Execute Just Works Owner Transfer Method
Onboarding device selects the oic.sec.oxm.jw owner transfer method and executes it.
1 POST /oic/sec/doxm {...,"oxmsel":0,...}
2 RSP 2.04
3 ClientHello(TLS_ECDHE_ANON_WITH_AES_128_CBC_SHA256)
4 HelloVerifyRequest(cookie)
5 ClientHello(TLS_ECDHE_ANON_WITH_AES_128_CBC_SHA256,cookie)
6 ServerHello(TLS_ECDHE_ANON_WITH_AES_128_CBC_SHA256) ServerKeyExchange(ECDH PublicKey + ECC Curve Param) ServerHelloDone()
7 ClientKeyExchange(ECDH PublicKey) ChangeCipherSpec + Finish
8 ChangeCipherSpec + Finish
On-boarding Device (UUID B0B0xxxx-...)
New Device (UUID A71C3xxx-...)

Figure 14 – A Just Works Owner Transfer Method

Step	Description
1, 2	The OBT notifies the Device that it selected the 'Just Works' method.
3 - 8	A DTLS session is established using anonymous Diffie-Hellman. Note: This method assumes the operator is aware of the potential for man-in-the-middle attack and has taken precautions to perform the method in a clean-room network.

Table 3 – A Just Works Owner Transfer Method Details

7.3.4.1 Security Considerations

Anonymous Diffie-Hellman key agreement is subject to a man-in-the-middle attacker. Use of this method presumes that both the OBT and the new device perform the 'just-works' method assumes onboarding happens in a relatively safe environment absent of an attack device.

This method doesn't have a trustworthy way to prove the device ID asserted is reliably bound to the device.

The new device should use a temporal device ID prior to transitioning to an owned device while it is considered a guest device to prevent privacy sensitive tracking. The device asserts a non-temporal device ID that could differ from the temporal value during the secure session in which owner transfer exchange takes place. The OBT will verify the asserted Device ID does not conflict with a Device ID already in use. If it is already in use the existing credentials are used to establish a secure session.

An un-owned Device that also has established device credentials might be an indication of a corrupted or compromised device.

7.3.5 Random PIN Based Owner Transfer Method

The Random PIN method establishes physical proximity between the new device and the OBT can prevent man-in-the-middle attacks. The Device generates a random number that is communicated to the OBT over an out-of-band channel. The definition of out-of-band communications channel is outside the scope of the definition of device owner transfer methods. The OBT and new Device use the PIN in a key exchange as evidence that someone authorized the transfer of ownership by having physical access to the new Device via the out-of-band-channel.

7.3.5.1 Random PIN Owner Transfer Sequence

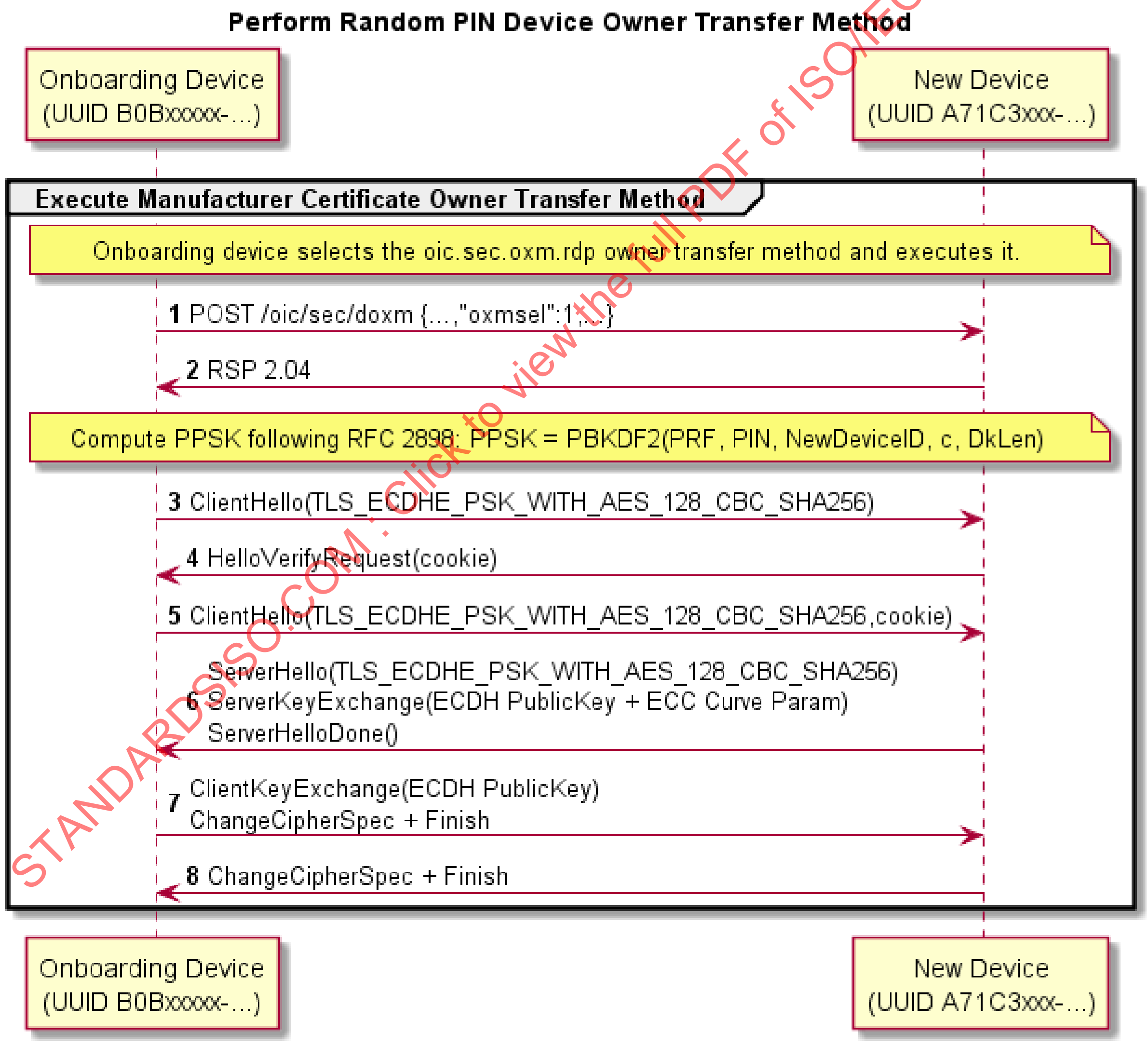

Perform Random PIN Device Owner Transfer Method
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Execute Manufacturer Certificate Owner Transfer Method
Onboarding device selects the oic.sec.oxm.rdp owner transfer method and executes it.
1 POST /oic/sec/doxm {...,"oxmsel":1,...}
2 RSP 2.04
Compute PPSK following RFC 2898: PPSK = PBKDF2(PRF, PIN, NewDeviceID, c, DkLen)
3 ClientHello(TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256)
4 HelloVerifyRequest(cookie)
5 ClientHello(TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,cookie)
6 ServerHello(TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256) ServerKeyExchange(ECDH PublicKey + ECC Curve Param) ServerHelloDone()
7 ClientKeyExchange(ECDH PublicKey) ChangeCipherSpec + Finish
8 ChangeCipherSpec + Finish
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

Figure 15 – Random PIN-based Owner Transfer Method

Step	Description
1, 2	The OBT notifies the Device that it selected the 'Random PIN' method.
3 - 8	A DTLS session is established using PSK-based Diffie-Hellman ciphersuite. The PIN is supplied as the PSK parameter. The PIN is randomly generated by the new device then communicated via an out-of-band channel that establishes proximal context between the new device and the OBT. The security principle is the attack device will be unable to intercept the PIN due to a lack of proximity.

Table 4 – Random PIN-based Owner Transfer Method Details

The random PIN-based device owner transfer method uses a pseudo-random function (PBKDF2) defined by RFC2898 and a PIN exchanged via an out-of-band method to generate a pre-shared key. The PIN-authenticated pre-shared key (PPSK) is supplied to TLS ciphersuites that accept a PSK.

PPSK = PBKDF2(PRF, PIN, Device ID, c, dkLen)

The PBKDF2 function has the following parameters:

- PRF – Uses the TLS 1.2 PRF defined by RFC5246.
- PIN – obtain via out-of-band channel.
- Device ID – UUID of the new device.
    - Use raw bytes as specified in RFC4122 section 4.1.2
- c – Iteration count initialized to 1000
- dkLen – Desired length of the derived PSK in octets.

7.3.5.2 Security Considerations

Security of the Random PIN mechanism depends on the entropy of the PIN. Using a PIN with insufficient entropy may allow a man-in-the-middle attack to recover any long-term credentials provisioned as a part of onboarding. In particular, learning provisioned symmetric key credentials, allows an attacker to masquerade as the onboarded device.

It is recommended that the entropy of the PIN be enough to withstand an online brute-force attack, 40 bits or more. For example, a 12-digit numeric PIN, or an 8-character alphanumeric (0-9a-z), or a 7 character case-sensitive alphanumeric PIN (0-9a-zA-Z). A man-in-the-middle attack (MITM) is when the attacker is active on the network and can intercept and modify messages between the OBT and device. In the MITM attack, the attacker must recover the PIN from the key exchange messages in "real time", i.e., before the peers time out and abort the connection attempt. Having recovered the PIN, he can complete the authentication step of key exchange. The guidance given here calls for a minimum of 40 bits of entropy, however, the assurance this provides depends on the resources available to the attacker. Given the paralleliziable nature of a brute force guessing attack, the attack enjoys a linear speedup as more cores/threads are added. A more conservative amount of entropy would be 64 bits. Since the Random PIN OTM requires using a DTLS ciphersuite that includes an ECDHE key exchange, the security of the Random PIN OTM is always at least equivalent to the security of the JustWorks OTM.

The Random PIN OTM also has an option to use PBKDF2 to derive key material from the PIN. The rationale is to increase the cost of a brute force attack, by increasing the cost of each guess in the attack by a tuneable amount (the number of PBKDF2 iterations). In theory, this is an effective way to reduce the entropy requirement of the PIN. Unfortunately, it is difficult to quantify the reduction, since an X-fold increase in time spent by the honest peers does not directly translate to an X-fold increase in time by the attacker. This asymmetry is because the attacker may use specialized implementations and hardware not available to honest peers. For this reason, when deciding how much entropy to use for a PIN, it is recommended that implementers assume PBKDF2 provides no security, and ensure the PIN has sufficient entropy.

The Random PIN device owner transfer method security depends on an assumption that a secure out-of-band method for communicating a randomly generated PIN from the new device to the OBT exists. If the OOB channel leaks some or the entire PIN to an attacker, this reduces the entropy of the PIN, and the attacks described above apply. The out-of-band mechanism should be chosen such that it requires proximity between the OBT and the new device. The attacker is assumed to not have compromised the out-of-band-channel. As an example OOB channel, the device may display a PIN to be entered into the OBT software. Another example is for the device to encode the PIN as a 2D barcode and display it for a camera on the OBT device to capture and decode.

7.3.6 Manufacturer Certificate Based Owner Transfer Method

The manufacturer certificate-based owner transfer method shall use a certificate embedded into the device by the manufacturer and may use a signed OBT, which determines the Trust Anchor between the device and the OBT.

When utilizing certificate-based ownership transfer, devices shall utilize asymmetric keys with certificate data to authenticate their identities with the OBT in the process of bringing a new device into operation on a user's network. The onboarding process involves several discrete steps:

1) Pre-on-board conditions
    a. The credential element of the Device's credential Resource (/oic/sec/cred) containing the manufacturer certificate shall be identified by the following properties:
        i. the subject Property shall refer to the Device
        ii. the credusage Property shall contain the string "oic.sec.cred.mfgcert" to indicate that the credential contains a manufacturer certificate
    b. The manufacturer certificate chain shall be contained in the identified credential element's publicdata Property with the optionaldata Property containing the Trust Anchor
    c. The device shall contain a unique and immutable ECC asymmetric key pair.
    d. If the device requires authentication of the OBT as part of ownership transfer, it is presumed that the OBT has been registered and has obtained a certificate for its unique and immutable ECC asymmetric key pair signed by the predetermined Trust Anchor.
    e. User has configured the OBT app with network access info and account info (if any).
2) The OBT shall authenticate the Device using ECDSA to verify the signature. Additionally the Device may authenticate the OBT to verify the OBT signature.
3) If authentication fails, the Device shall indicate the reason for failure and return to the Ready for OTM state. If authentication succeeds, the device and OBT shall establish an encrypted link in accordance with the negotiated cipher suite.

7.3.6.1 Certificate Profiles

Within the Device PKI, the following format shall be used for the `subject` within the certificates. It is anticipated that there may be N distinct roots for scalability and failover purposes. The vendor creating and operating root will be approved by the OCF based on due process described in Certificate Policy (CP) document and appropriate RFP documentation. Each root may issue one or more DEV CAs, which in turn issue Manufacturer DEV CAs to individual manufacturers. A manufacturer may decide to request for more than one Manufacturer CAs. Each Manufacturer CA issues one or more Device Sub-CAs (up to M) and issues one or more OCSP responders (up to O). For now we can assume that revocation checking for any CA certificates is handled by CRLs issued by the higher level CAs.

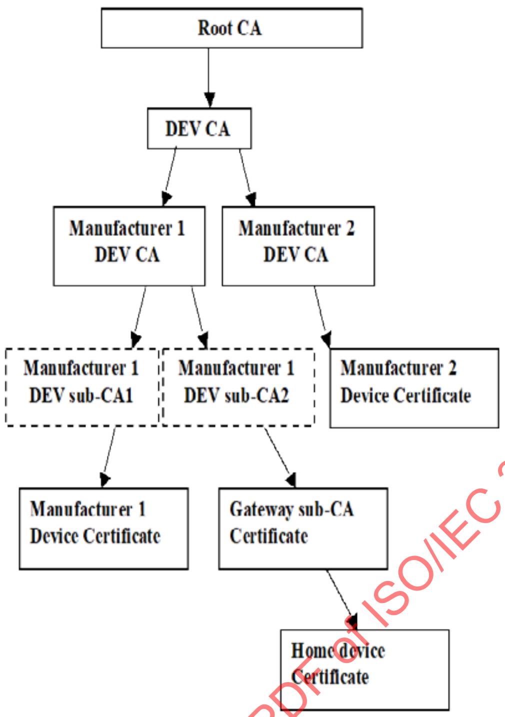

Root CA
DEV CA
Manufacturer 1 DEV CA
Manufacturer 2 DEV CA
Manufacturer 1 DEV sub-CA1
Manufacturer 1 DEV sub-CA2
Manufacturer 2 Device Certificate
Manufacturer 1 Device Certificate
Gateway sub-CA Certificate
Home device Certificate

Figure 16 – Manufacturer Certificate Hierarchy

- Root CA: C=<country where the root was created>, O=<name of root CA vendor>, OU=OCF Root CA, CN=OCF (R) Device Root-CA<n>
- DEV CA: C=<country for the DEV CA>, O=<name of root CA vendor>, OU=OCF DEV CA, CN=<name of DEV CA defined by root CA vendor>
- Manufacturer DEV CA: C=<country where Manufacturer DEV CA is registered>, O=<name of root CA vendor>, OU=OCF Manufacturer DEV CA, CN=<name defined by manufacturer><m>
- Device Sub-CA: C=<country device sub-CA>, O=<name of root CA vendor>, OU=OCF Manufacturer Device sub-CA, OU=<defined by Manufacturer>, CN=<defined by manufacturer>
- For Device Sub-CA Level OCSP Responder: C=<country of device Sub-CA>, O=<name of root CA vendor>, OU=OCF Manufacturer OCSP Responder <o>, CN=<name defined by CA vendor >
- Device cert: C=<country>, O=<manufacturer>, OU=Device, CN=<device Type><single space (i.e., " ")><device model name>
    - o The following optional naming elements MAY be included between the OU=OCF (R) Devices and CN= naming elements. They MAY appear in any order: OU=chipsetID: <chipsetID>, OU=<device type>, OU=<device model name> OU=<mac address> OU=<device security profile>
- Gateway Sub-CA: C=<country>, O=<manufacturer>, OU=<manufacture name> Gateway sub-CA, CN=<name defined by manufacturer>, <unique Gateway identifier generated with UAID method>
- Home Device Cert: C=<country>, O=<manufacturer>, OU=Non-Device cert, OU=<Gateway UAID>, CN=<device Typle>

Technical Note regarding Gateway Sub-CA: If a manufacturer decides to allow its Gateways to act as Gateway Sub-CA, it needs to accommodate this by setting the proper value on path-length-constraint value within the Device Sub-CA certificate, to allow the latter sub-CA to issue CA certificates to Gateway Sub-CAs. Given that the number of Gateway Sub-CAs can be very large a numbering scheme should be used for Gateway Sub-CA ID and given the Gateway does have public key pair, UAID algorithm shall be used to calculate the gateway identifier using a hash of gateway public key and inserted inside subject field of Gateway Sub-CA certificate.

A separate Device Sub-CA shall be used to generate Gateway Sub-CA certificates. This Device Sub-CA shall not be used for issuance of non-Gateway device certificates.

CRLs including Gatway Sub-CA certificates shall be issued on monthly basis, rather than quarterly basis to avoid potentially large liabilities related to Gateway Sub-CA compromise.

Device certificates issued by Gateway Sub-CA shall include an OU=Non-Device cert, to indicate that they are not issued by an OCF governed CA.

When the naming element is DirectoryString (i.e., O=, OU=) either PrintableString or UTF8String shall be used. The following determines which choice is used:

- PrintableString only if it is limited to the following subset of US ASCII characters (as required by ASN.1):
  A, B, …, Z
  a, b, …, z
  0, 1, …9,
  (space) ‘ ( ) + , - . / : = ?
- UTF8String for all other cases, e.g., subject name attributes with any other characters or for international character sets.

A CVC CA is used by a trusted organization to issue CVC code signing certificates to software providers, system administrators, or other entities that will sign software images for the Devices. A CVC CA shall not sign and issue certificates for any specialization other than code signing. In other words, the CVC CA shall not sign and issue certificates that belong to any branches other than the CVC branch.

7.3.6.2 Certificate Owner Transfer Sequence Security Considerations

In order for full, mutual authentication to occur between the device and the OBT, both the device and OBT must be able to trace back to a mutual Trust Anchor or Certificate Authority. This implies that OCF may need to obtain services from a Certificate Authority (e.g. Symantec, Verisign, etc.) to provide ultimate Trust Anchors from which all subsequent OCF Trust Anchors are derived.

The OBT shall authenticate the device during onboarding. However, the device is not required to authenticate the OBT due to potential resource constraints on the device.

In the case where the Device does NOT authenticate the OBT software, there is the possibility of malicious OBT software unwittingly deployed by users, or maliciously deployed by an adversary, which can compromise network access credentials and/or personal information.

7.3.6.3 Manufacturer Certificate Based Owner Transfer Method Sequence

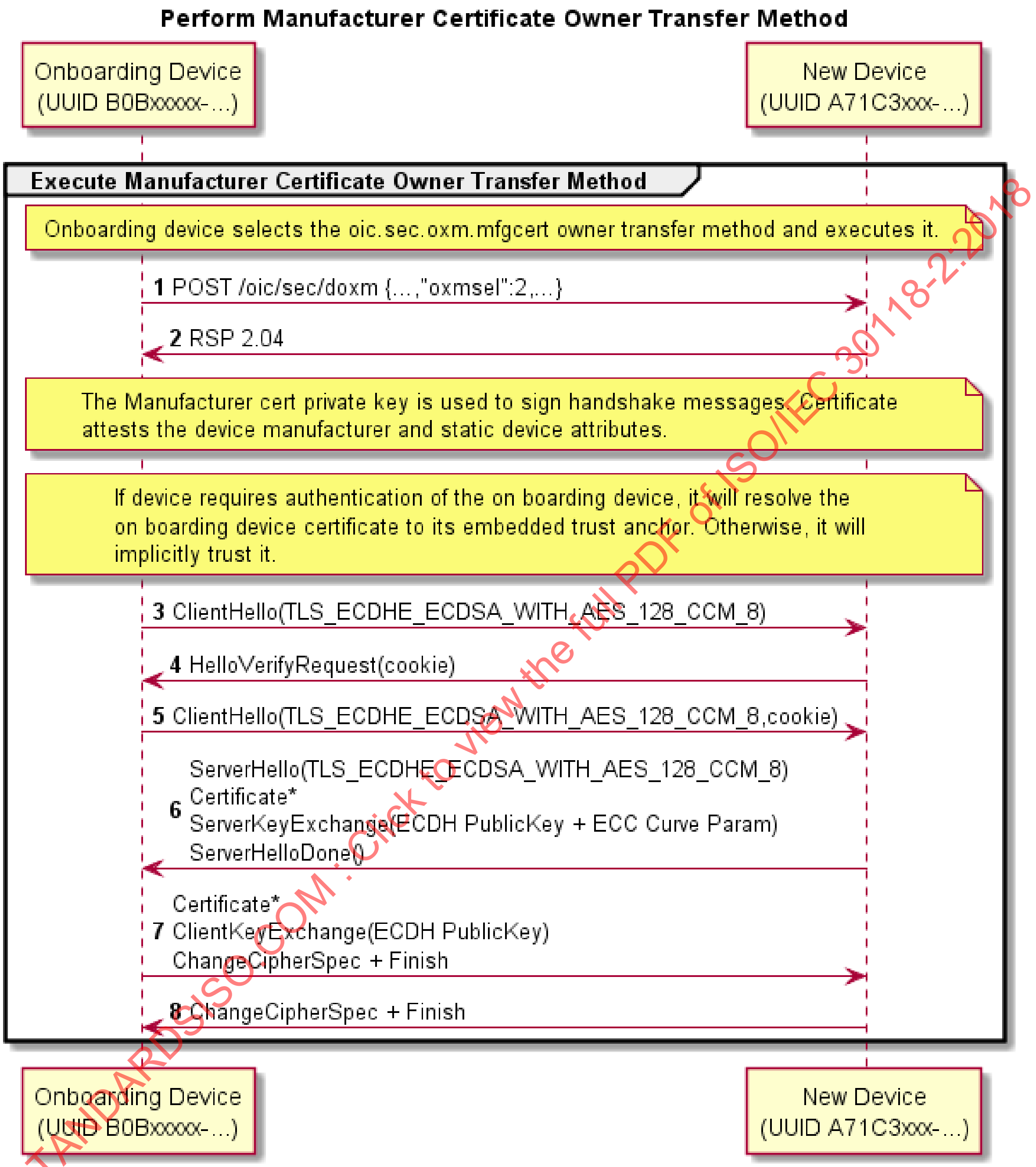

Perform Manufacturer Certificate Owner Transfer Method
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Execute Manufacturer Certificate Owner Transfer Method
Onboarding device selects the oic.sec.oxm.mfgcert owner transfer method and executes it.
1 POST /oic/sec/doxm {...,"oxmsel":2,...}
2 RSP 2.04
The Manufacturer cert private key is used to sign handshake messages. Certificate attests the device manufacturer and static device attributes.
If device requires authentication of the on boarding device, it will resolve the on boarding device certificate to its embedded trust anchor. Otherwise, it will implicitly trust it.
3 ClientHello(TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8)
4 HelloVerifyRequest(cookie)
5 ClientHello(TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8,cookie)
6 ServerHello(TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8) Certificate* ServerKeyExchange(ECDH PublicKey + ECC Curve Param) ServerHelloDone()
7 Certificate* ClientKeyExchange(ECDH PublicKey) ChangeCipherSpec + Finish
8 ChangeCipherSpec + Finish
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

Figure 17 – Manufacturer Certificate Based Owner Transfer Method Sequence

Step	Description
1, 2	The OBT notifies the Device that it selected the 'Manufacturer Certificate' method.
3 - 8	A DTLS session is established using the device's manufacturer certificate and optional OBT certificate. The device's manufacturer certificate may contain data attesting to the Device hardening and security properties.

Table 5 – Manufacturer Certificate Based Owner Transfer Method Details

7.3.6.4 Security Considerations

The manufacturer certificate private key is embedded in the Platform with a sufficient degree of assurance that the private key cannot be compromised.

The Platform manufacturer issues the manufacturer certificate and attests the private key protection mechanism.

The manufacturer certificate defines its uniqueness properties.

There may be multiple Device instances hosted by a Platform containing a single manufacturer certificate

7.3.7 Vendor Specific Owner Transfer Methods

The OCF anticipates situations where a vendor will need to implement an owner transfer method that accommodates manufacturing or Device constraints. The Device owner transfer method resource is extensible for this purpose. Vendor-specific owner transfer methods must adhere to a set of conventions that all owner transfer methods follow.

- The OBT must determine which credential types are supported by the Device. This is accomplished by querying the Device's /oic/sec/doxm Resource to identify supported credential types.
- The OBT provisions the Device with OC(s).
- The OBT supplies the Device ID and credentials for subsequent access to the OBT.
- The OBT will supply second carrier settings sufficient for accessing the owner's network subsequent to ownership establishment.
- The OBT may perform additional provisioning steps but must not invalidate provisioning tasks to be performed by a bootstrap or security service.

7.3.7.1 Vendor-specific Owner Transfer Sequence Example

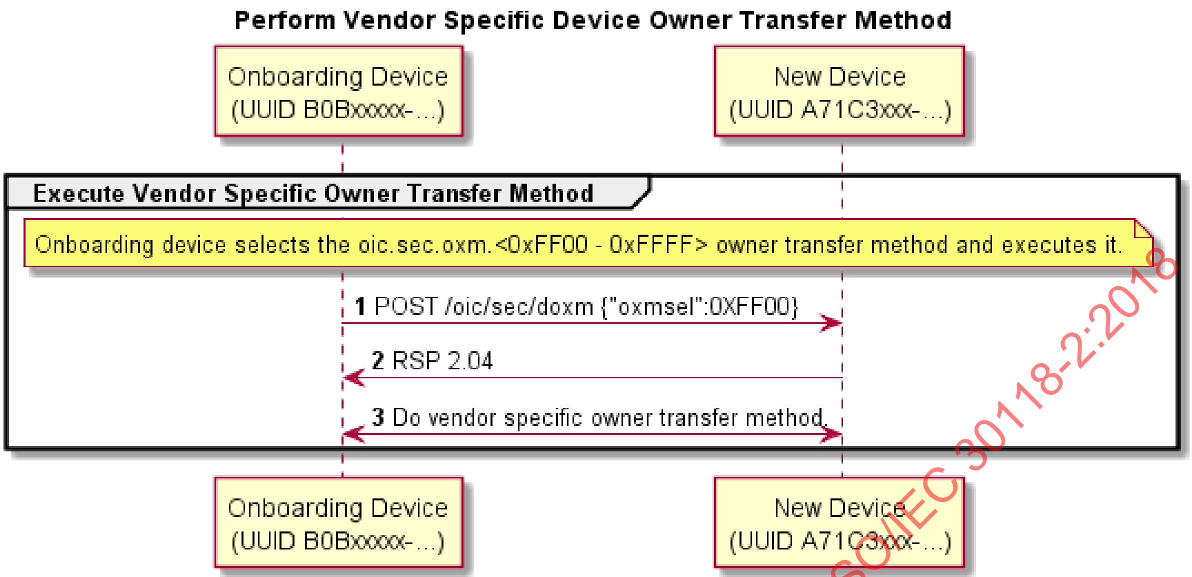

Perform Vendor Specific Device Owner Transfer Method
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Execute Vendor Specific Owner Transfer Method
Onboarding device selects the oic.sec.oxm.<0xFF00 - 0xFFFF> owner transfer method and executes it.
1 POST /oic/sec/doxm {"oxmsel":0XFF00}
2 RSP 2.04
3 Do vendor specific owner transfer method.
Onboarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

Figure 18 – Vendor-specific Owner Transfer Sequence

Step	Description
1, 2	The OBT selects a vendor-specific owner transfer method.
3	The vendor-specific owner transfer method is applied

Table 6 – Vendor-specific Owner Transfer Details

7.3.7.2 Security Considerations

The vendor is responsible for considering security threats and mitigation strategies.

7.3.8 Establishing Owner Credentials

Once the OBT and the new Device have authenticated and established an encrypted connection using one of the defined OTM methods.

Owner credentials may consist of certificates signed by the OBT or other authority, user network access information, provisioning functions, shared keys, or Kerberos tickets.

The OBT might then provision the new Device with additional credentials for Device management and Device-to-Device communications. These credentials may consist of certificates with signatures, UAID based on the Device public key, PSK, etc.

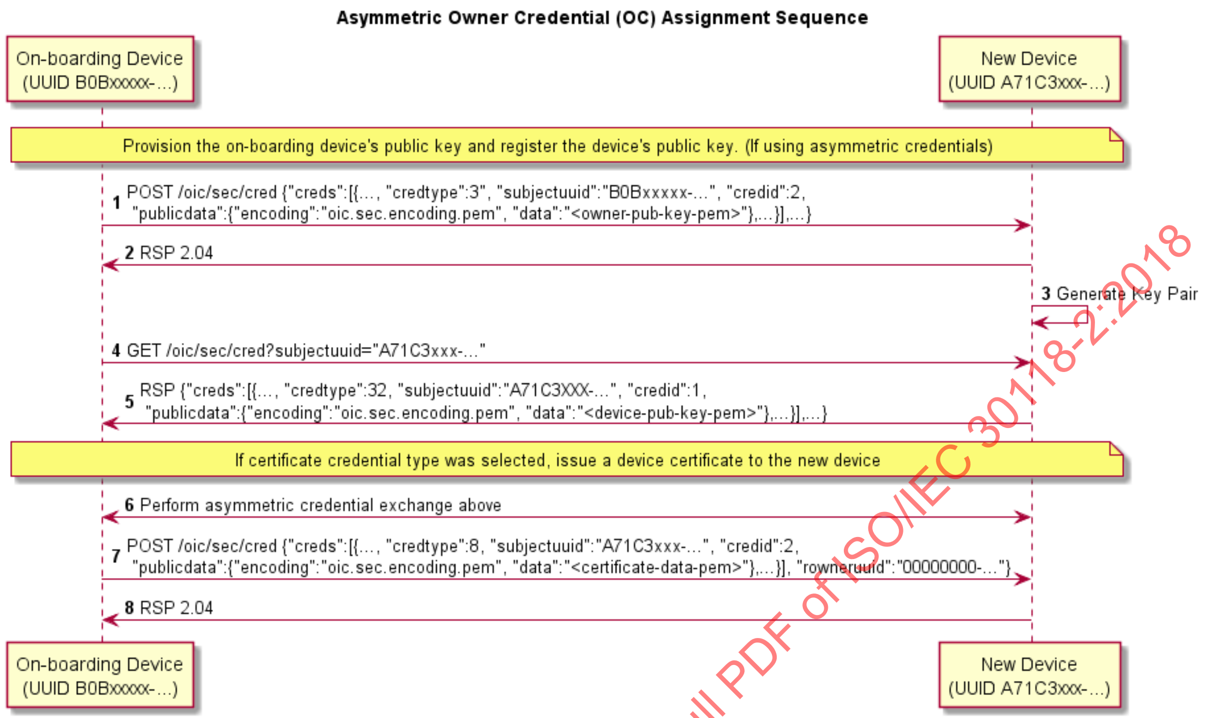
The steps for establishing Device's owner credentials (OC) are detailed below:

a. The OBT shall establish the Device ID and Device owner uuid - Figure 19

b. The OBT then establishes Device’s owner credentials (OC) - Figure 20. This can be either:

 i. Symmetric credential - Figure 21

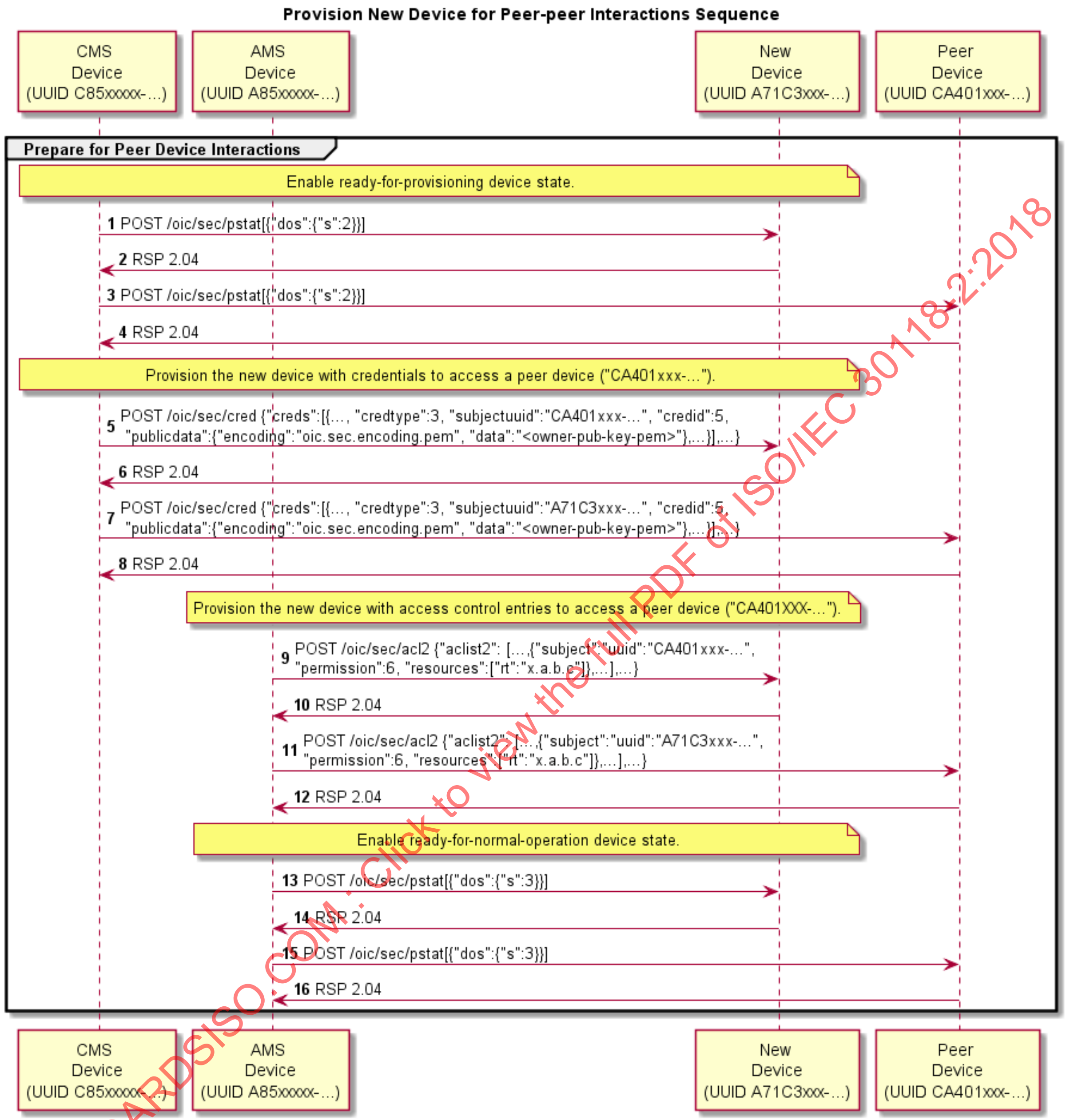
Asymmetric credential -


Asymmetric Owner Credential (OC) Assignment Sequence
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Provision the on-boarding device's public key and register the device's public key. (If using asymmetric credentials)
1 POST /oic/sec/cred {"creds":[{..., "credtype":"3", "subjectuuid":"B0Bxxxxx-...", "credid":2, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
2 RSP 2.04
3 Generate Key Pair
4 GET /oic/sec/cred?subjectuuid="A71C3xxx-..."
5 RSP {"creds":[{..., "credtype":32, "subjectuuid":"A71C3XXX-...", "credid":1, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<device-pub-key-pem>"},...}],...}
If certificate credential type was selected, issue a device certificate to the new device
6 Perform asymmetric credential exchange above
7 POST /oic/sec/cred {"creds":[{..., "credtype":8, "subjectuuid":"A71C3xxx-...", "credid":2, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<certificate-data-pem>"},...}], "rowneruuid":"00000000-..."}
8 RSP 2.04
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

ii. Figure 22

c. Configure Device services. - Figure 23

Configure Device for peer to peer interaction -


Provision New Device for Peer-peer Interactions Sequence
CMS Device (UUID C85xxxxx-...)
AMS Device (UUID A85xxxxx-...)
New Device (UUID A71C3xxx-...)
Peer Device (UUID CA401xxx-...)
Prepare for Peer Device Interactions
Enable ready-for-provisioning device state.
1 POST /oic/sec/pstat[{"dos":{"s":2}}]
2 RSP 2.04
3 POST /oic/sec/pstat[{"dos":{"s":2}}]
4 RSP 2.04
Provision the new device with credentials to access a peer device ("CA401xxx-...").
5 POST /oic/sec/cred {"creds":[{..., "credtype":3, "subjectuuid":"CA401xxx-...", "credid":5, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
6 RSP 2.04
7 POST /oic/sec/cred {"creds":[{..., "credtype":3, "subjectuuid":"A71C3xxx-...", "credid":5, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
8 RSP 2.04
Provision the new device with access control entries to access a peer device ("CA401XXX-...").
9 POST /oic/sec/acl2 {"aclist2": [...,{"subject":"uuid":"CA401xxx-...", "permission":6, "resources":["rt":"x.a.b.c"]},...],...}
10 RSP 2.04
11 POST /oic/sec/acl2 {"aclist2": [...,{"subject":"uuid":"A71C3xxx-...", "permission":6, "resources":["rt":"x.a.b.c"]},...],...}
12 RSP 2.04
Enable ready-for-normal-operation device state.
13 POST /oic/sec/pstat[{"dos":{"s":3}}]
14 RSP 2.04
15 POST /oic/sec/pstat[{"dos":{"s":3}}]
16 RSP 2.04
CMS Device (UUID C85xxxxx-...)
AMS Device (UUID A85xxxxx-...)
New Device (UUID A71C3xxx-...)
Peer Device (UUID CA401xxx-...)

d. Figure 24

These credentials may consist of certificates signed by the OBT or other authority, user network access information, provisioning functions, shared keys, or Kerberos tickets.

The OBT might then provision the new Device with additional credentials for Device management and Device-to-Device communications. These credentials may consist of certificates with signatures, UAID based on the Device public key, PSK, etc.

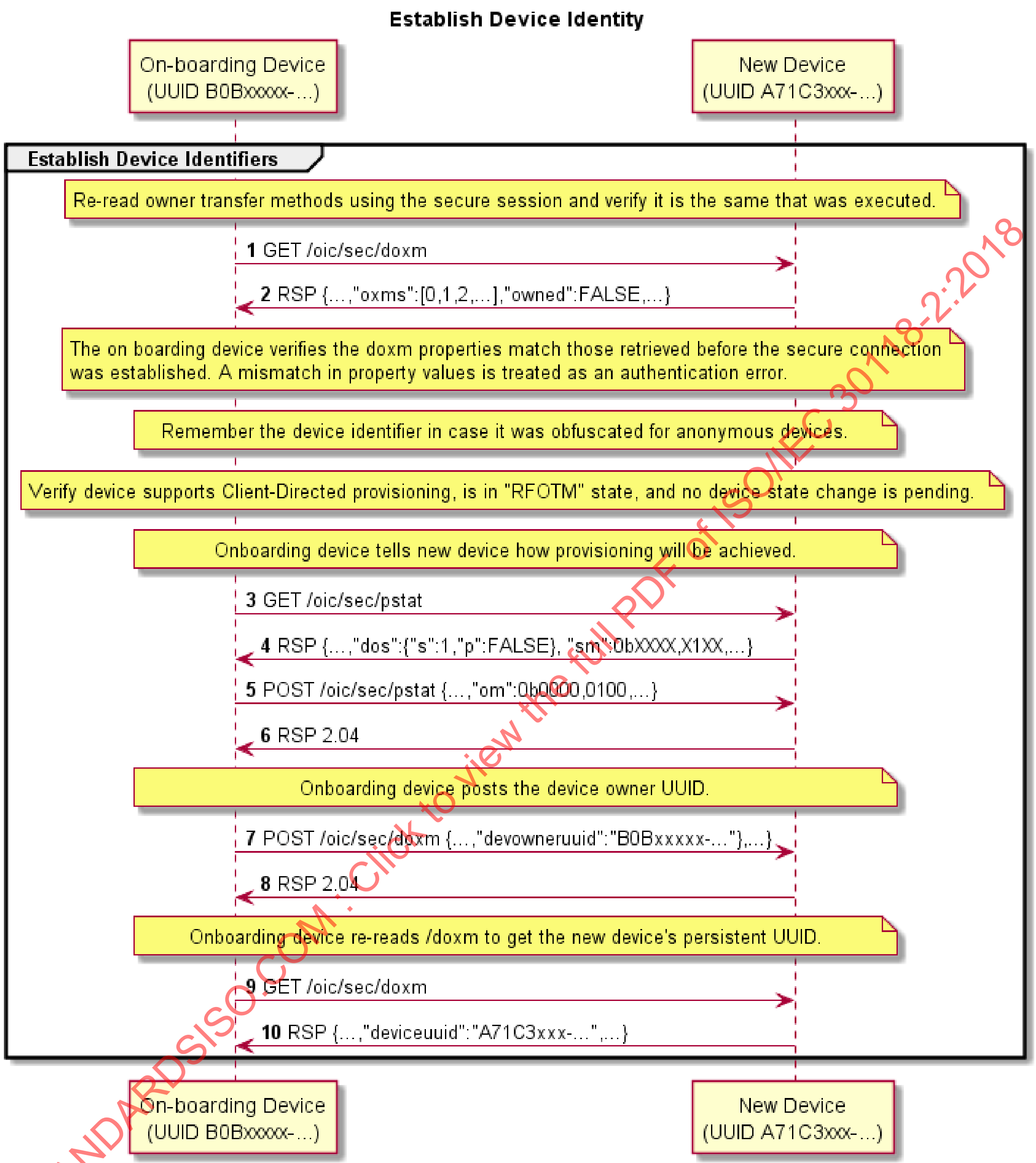
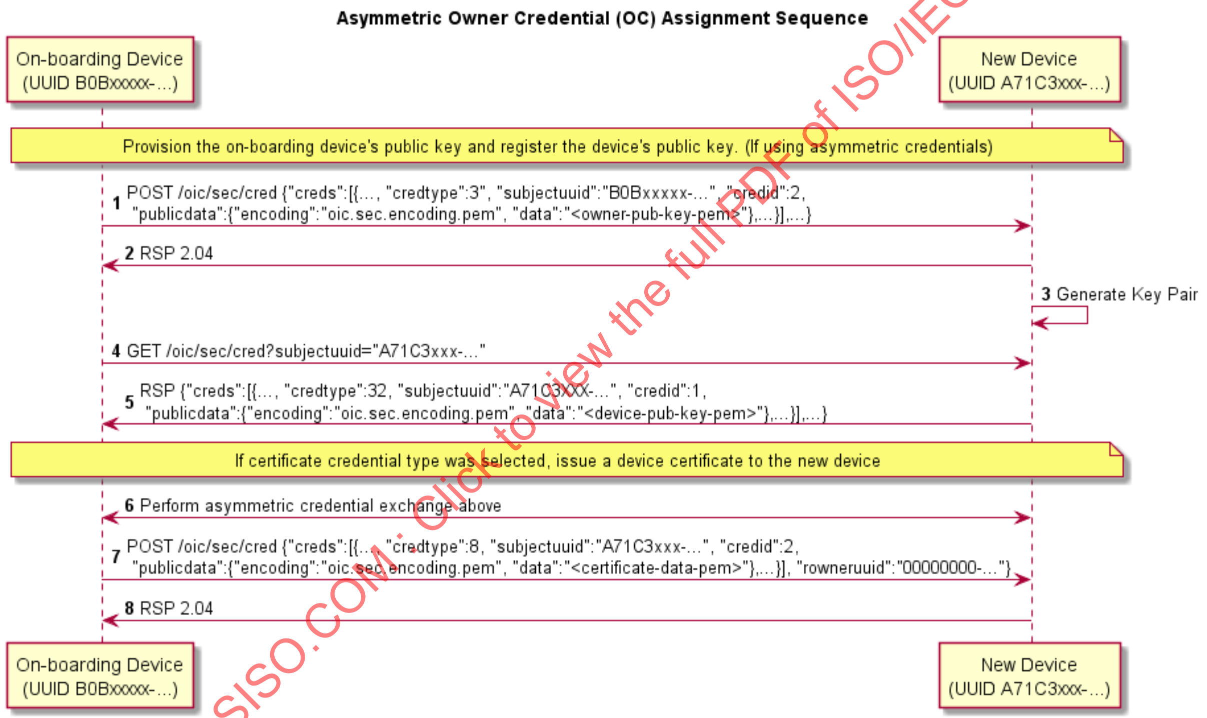

Establish Device Identity
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Establish Device Identifiers
Re-read owner transfer methods using the secure session and verify it is the same that was executed.
1 GET /oic/sec/doxm
2 RSP {...,"oxms":[0,1,2,...],"owned":FALSE,...}
The on boarding device verifies the doxm properties match those retrieved before the secure connection was established. A mismatch in property values is treated as an authentication error.
Remember the device identifier in case it was obfuscated for anonymous devices.
Verify device supports Client-Directed provisioning, is in "RFOTM" state, and no device state change is pending.
Onboarding device tells new device how provisioning will be achieved.
3 GET /oic/sec/pstat
4 RSP {...,"dos":{"s":1,"p":FALSE}, "sm":0bXXXX,X1XX,...}
5 POST /oic/sec/pstat {...,"om":0b0000,0100,...}
6 RSP 2.04
Onboarding device posts the device owner UUID.
7 POST /oic/sec/doxm {...,"devowneruuid":"B0Bxxxxx-..."},...}
8 RSP 2.04
Onboarding device re-reads /doxm to get the new device's persistent UUID.
9 GET /oic/sec/doxm
10 RSP {...,"deviceuuid":"A71C3xxx-...",...}
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

Figure 19 - Establish Device Identity Flow

Step	Description
1, 2	The OBT obtains the doxm properties again, using the secure session. It verifies that these properties match those retrieved before the authenticated connection. A mismatch in parameters is treated as an authentication error.
3, 4	The OBT queries to determine if the Device is operationally ready to transfer Device ownership.
5, 6	The OBT asserts that it will follow the Client provisioning convention.
7, 8	The OBT asserts itself as the owner of the new Device by setting the Device ID to its ID.
9, 10	The OBT obtains doxm properties again, this time Device returns new Device persistant UUID.

Table 7 - Establish Device Identity Details

group See


Asymmetric Owner Credential (OC) Assignment Sequence
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Provision the on-boarding device's public key and register the device's public key. (If using asymmetric credentials)
3 Generate Key Pair
If certificate credential type was selected, issue a device certificate to the new device
6 Perform asymmetric credential exchange above
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

```
1 POST /oic/sec/cred {"creds":[{..., "credtype":3", "subjectuuid":"B0Bxxxxx-...", "credid":2,
  "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
2 RSP 2.04
4 GET /oic/sec/cred?subjectuuid="A71C3xxx-..."
5 RSP {"creds":[{..., "credtype":32, "subjectuuid":"A71C3XXX-...", "credid":1,
  "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<device-pub-key-pem>"},...}],...}
7 POST /oic/sec/cred {"creds":[{..., "credtype":8, "subjectuuid":"A71C3xxx-...", "credid":2,
  "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<certificate-data-pem>"},...}], "rowneruuid":"00000000-..."}
8 RSP 2.04
```

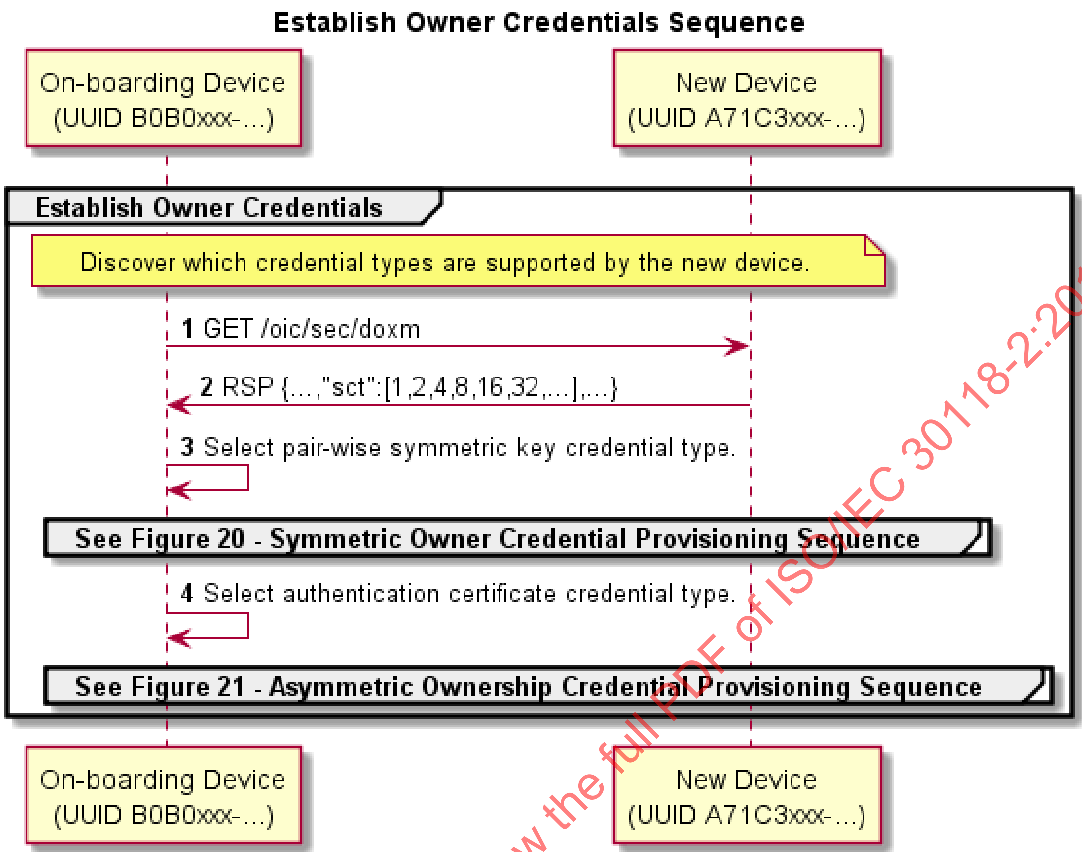

Establish Owner Credentials Sequence
On-boarding Device (UUID B0B0xxx-...)
New Device (UUID A71C3xxx-...)
Establish Owner Credentials
Discover which credential types are supported by the new device.
1 GET /oic/sec/doxm
2 RSP {...,"sct":[1,2,4,8,16,32,...],...}
3 Select pair-wise symmetric key credential type.
See Figure 20 - Symmetric Owner Credential Provisioning Sequence
4 Select authentication certificate credential type.
See Figure 21 - Asymmetric Ownership Credential Provisioning Sequence
On-boarding Device (UUID B0B0xxx-...)
New Device (UUID A71C3xxx-...)

Figure 20 – Owner Credential Selection Provisioning Sequence

Step	Description
1, 2	The OBT obtains the doxm properties to check ownership transfer mechanism supported on the new Device.
3, 4	The OBT uses selected credential type for ownership provisioning.

Table 8 - Owner Credential Selection Details

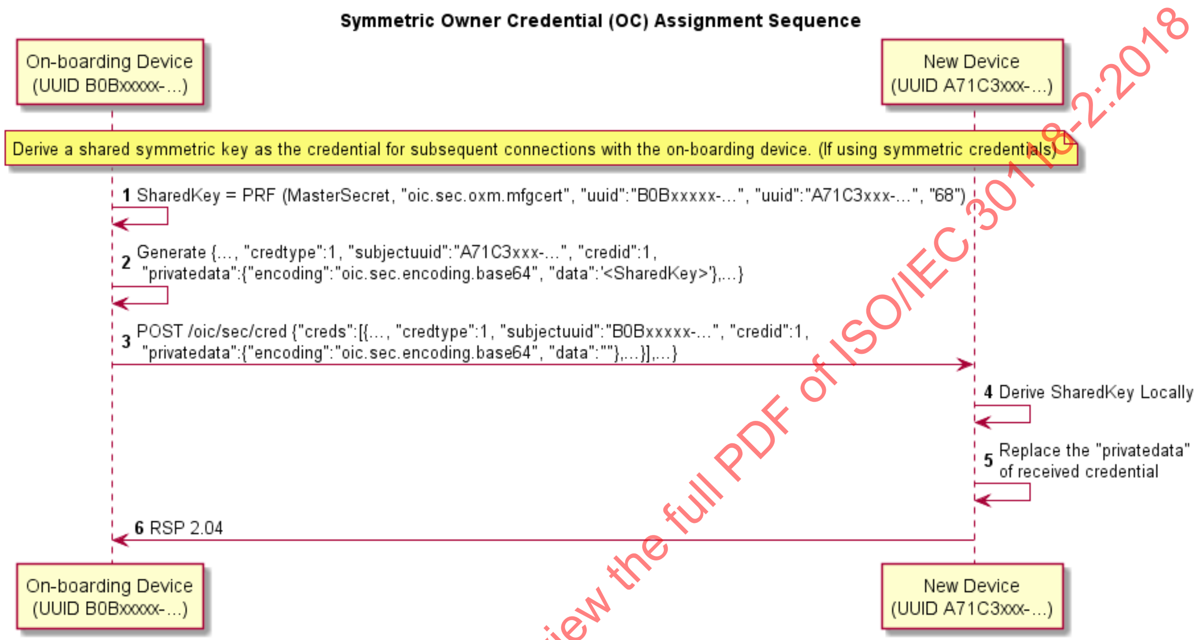

Symmetric Owner Credential (OC) Assignment Sequence
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Derive a shared symmetric key as the credential for subsequent connections with the on-boarding device. (If using symmetric credentials)
1 SharedKey = PRF (MasterSecret, "oic.sec.oxm.mfgcert", "uuid":"B0Bxxxxx-...", "uuid":"A71C3xxx-...", "68")
2 Generate {..., "credtype":1, "subjectuuid":"A71C3xxx-...", "credid":1, "privatedata":{"encoding":"oic.sec.encoding.base64", "data":'<SharedKey>'},...}
3 POST /oic/sec/cred {"creds":[{..., "credtype":1, "subjectuuid":"B0Bxxxxx-...", "credid":1, "privatedata":{"encoding":"oic.sec.encoding.base64", "data":""},...}],...}
4 Derive SharedKey Locally
5 Replace the "privatedata" of received credential
6 RSP 2.04
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

Figure 21 - Symmetric Owner Credential Provisioning Sequence

Step	Description
1, 2	The OBT uses a pseudo-random-function (PRF), the master secret resulting from the DTLS handshake, and other information to generate a symmetric key credential resource property - SharedKey.
3	The OBT creates a credential resource property set based on SharedKey and then sends the resource property set to the new Device with empty "privatedata" property value.
4, 5	The new Device locally generates the SharedKey and updates it to the "privatedata" property of the credential resource property set.
6	The new Device sends a success message.

Table 9 - Symmetric Owner Credential Assignment Details

In particular, if the OBT selects symmetric owner credentials:

- The OBT shall generate a Shared Key using the SharedKey Credential Calculation method described in Section 7.3.2.
- The OBT shall send an empty key to the new Device’s /oic/sec/cred Resource, identified as a symmetric pair-wise key.
- Upon receipt of the OBT’s symmetric owner credential, the new Device shall independently generate the Shared Key using the SharedKey Credential Calculation method described in Section 7.3.2 and store it with the owner credential.

- The new Device shall use the Shared Key owner credential(s) stored via the /oic/sec/cred Resource to authenticate the owner during subsequent connections.

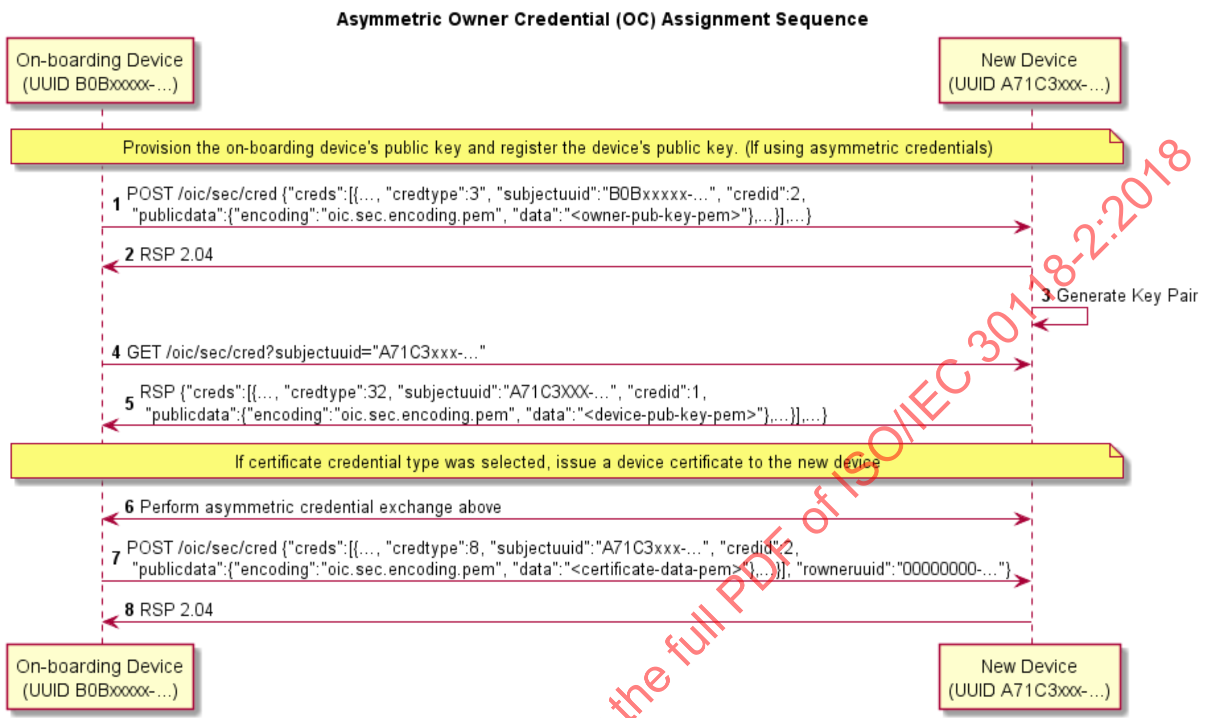

Asymmetric Owner Credential (OC) Assignment Sequence
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)
Provision the on-boarding device's public key and register the device's public key. (If using asymmetric credentials)
1 POST /oic/sec/cred {"creds":[{..., "credtype":"3", "subjectuuid":"B0Bxxxxx-...", "credid":2, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
2 RSP 2.04
3 Generate Key Pair
4 GET /oic/sec/cred?subjectuuid="A71C3xxx-..."
5 RSP {"creds":[{..., "credtype":32, "subjectuuid":"A71C3XXX-...", "credid":1, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<device-pub-key-pem>"},...}],...}
If certificate credential type was selected, issue a device certificate to the new device
6 Perform asymmetric credential exchange above
7 POST /oic/sec/cred {"creds":[{..., "credtype":8, "subjectuuid":"A71C3xxx-...", "credid":2, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<certificate-data-pem>"},...}], "rowneruuid":"00000000-..."}
8 RSP 2.04
On-boarding Device (UUID B0Bxxxxx-...)
New Device (UUID A71C3xxx-...)

Figure 22 - Asymmetric Ownership Credential Provisioning Sequence

Step	Description
If an asymmetric or certificate owner credential type was selected by the OBT	
1, 2	The OBT creates an asymmetric type credential Resource property set with its pubic key (OC) to the new Device. It may be used subsequently to authenticate the OBT. The new device creates a credential Resource property set based on the public key generated.
3	The new Device creates an asymmetric key pair.
4, 5	The OBT reads the new Device's asymmetric type credential Resource property set generated at step 25. It may be used subsequently to authenticate the new Device.
If certificate owner credential type is selected by the OBT	
6-8	The steps for creating an asymmetric credential type are performed. In addition, the OBT instantiates a newly-created certificate (or certificate chain) on the new Device.

Table 10 – Asymmetric Owner Credential Assignment Details

If the OBT selects asymmetric owner credentials:

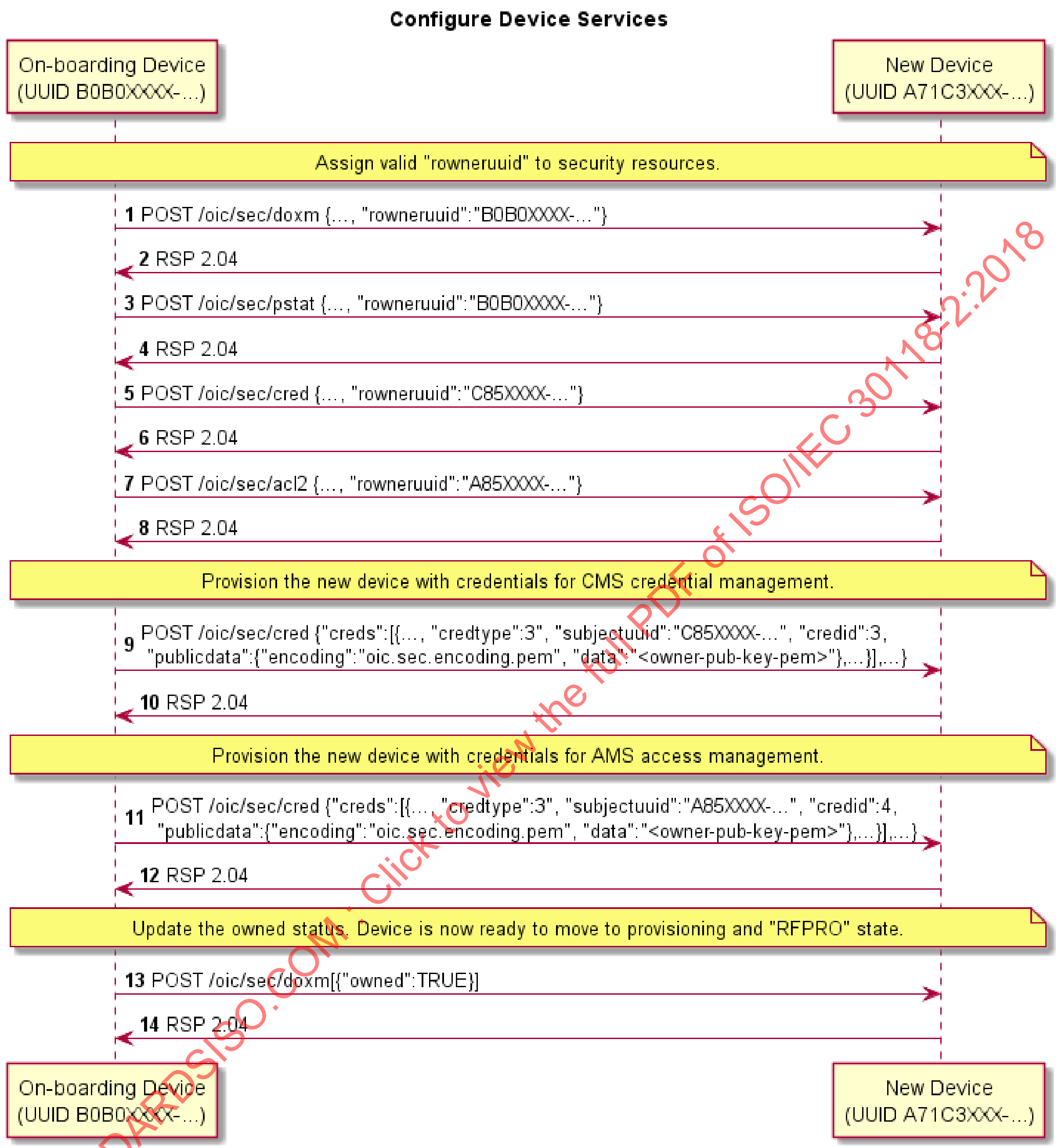
- The OBT shall add its public key to the new Device's /oic/sec/cred Resource, identified as an Asymmetric Encryption Key.

- The OBT shall query the /oic/sec/cred Resource from the new Device, supplying the new Device's UUID via the SubjectID query parameter. In response, the new Device shall return the public Asymmetric Encryption Key, which the OBT shall retain for future owner authentication of the new Device.

If the OBT selects certificate owner credentials:

- The OBT shall create a certificate or certificate chain with the leaf certificate containing the public key returned by the new Device, signed by a mutually-trusted CA, and complying with the Certificate Credential Generation requirements defined in Section 7.3.3.

- The OBT shall add the newly-created certificate chain to the /oic/sec/cred Resource, identified as an Asymmetric Signing Key with Certificate.


Configure Device Services
On-boarding Device (UUID B0B0XXXX-...)
New Device (UUID A71C3XXX-...)
Assign valid "rowneruuid" to security resources.
Provision the new device with credentials for CMS credential management.
Provision the new device with credentials for AMS access management.
Update the owned status. Device is now ready to move to provisioning and "RFPRO" state.
On-boarding Device (UUID B0B0XXXX-...)
New Device (UUID A71C3XXX-...)

```
1 POST /oic/sec/doxm {..., "rowneruuid":"B0B0XXXX-..."}
2 RSP 2.04
3 POST /oic/sec/pstat {..., "rowneruuid":"B0B0XXXX-..."}
4 RSP 2.04
5 POST /oic/sec/cred {..., "rowneruuid":"C85XXXX-..."}
6 RSP 2.04
7 POST /oic/sec/acl2 {..., "rowneruuid":"A85XXXX-..."}
8 RSP 2.04
9 POST /oic/sec/cred {"creds":[{..., "credtype":3", "subjectuuid":"C85XXXX-...", "credid":3,
  "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
10 RSP 2.04
11 POST /oic/sec/cred {"creds":[{..., "credtype":3", "subjectuuid":"A85XXXX-...", "credid":4,
   "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
12 RSP 2.04
13 POST /oic/sec/doxm[{"owned":TRUE}]
14 RSP 2.04
```

Figure 23 - Configure Device Services

Step	Description
1 - 8	The OBT assigns rowneruuid for different SVRs.
9 - 10	Provision the new Device with credentials for CMS
11 - 12	Provision the new Device with credentials for AMS
13 - 14	Update the oic.sec.doxm.owned to TRUE. Device is ready to move to provision and RFPRO state.

Table 11 - Configure Device Services Detail

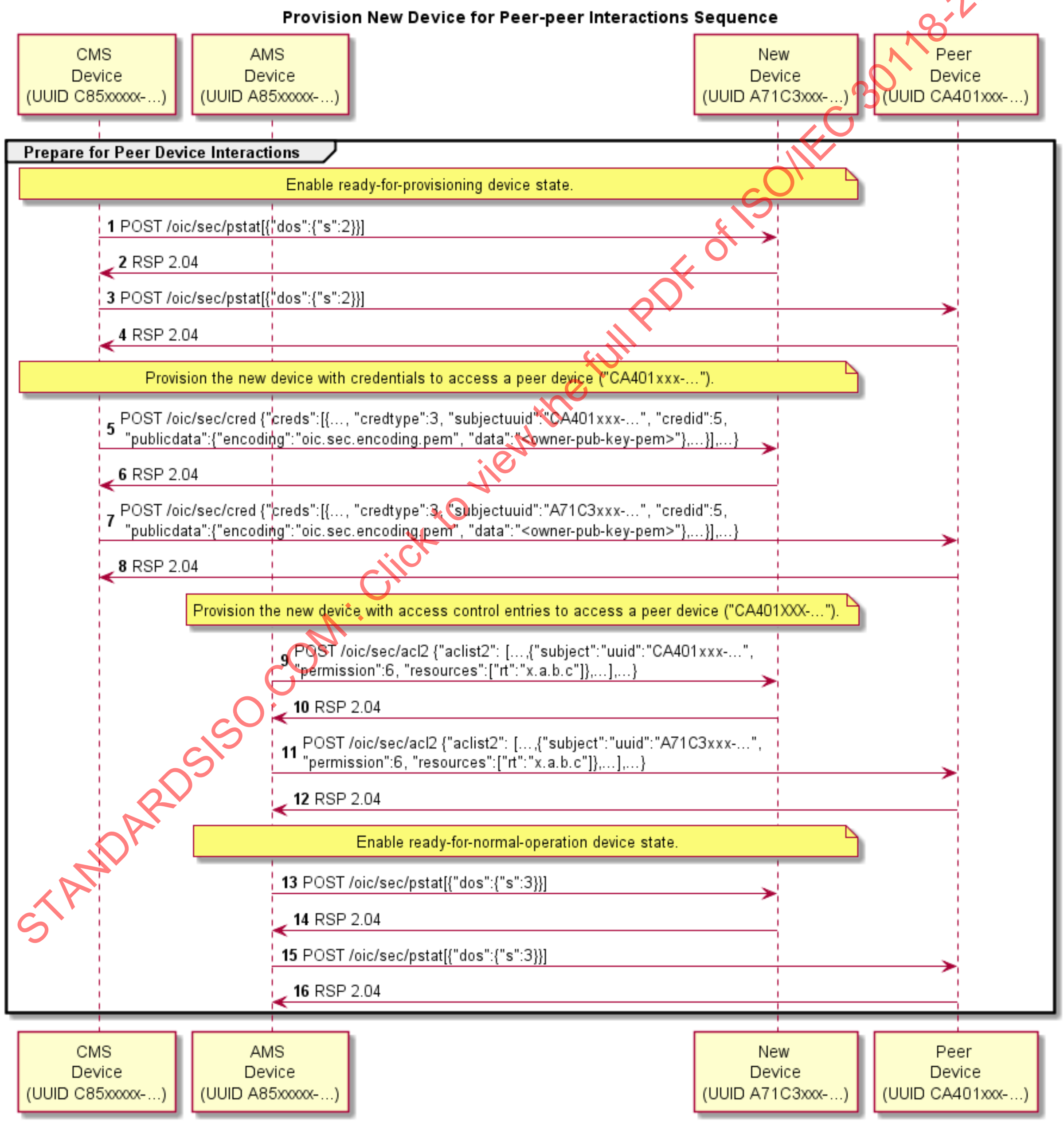

Provision New Device for Peer-peer Interactions Sequence
CMS Device (UUID C85xxxxx-...)
AMS Device (UUID A85xxxxx-...)
New Device (UUID A71C3xxx-...)
Peer Device (UUID CA401xxx-...)
Prepare for Peer Device Interactions
Enable ready-for-provisioning device state.
1 POST /oic/sec/pstat[{"dos":{"s":2}}]
2 RSP 2.04
3 POST /oic/sec/pstat[{"dos":{"s":2}}]
4 RSP 2.04
Provision the new device with credentials to access a peer device ("CA401xxx-...").
5 POST /oic/sec/cred {"creds":[{..., "credtype":3, "subjectuuid":"CA401xxx-...", "credid":5, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
6 RSP 2.04
7 POST /oic/sec/cred {"creds":[{..., "credtype":3, "subjectuuid":"A71C3xxx-...", "credid":5, "publicdata":{"encoding":"oic.sec.encoding.pem", "data":"<owner-pub-key-pem>"},...}],...}
8 RSP 2.04
Provision the new device with access control entries to access a peer device ("CA401XXX-...").
9 POST /oic/sec/acl2 {"aclist2": [...,{"subject":"uuid":"CA401xxx-...", "permission":6, "resources":["rt":"x.a.b.c"]},...],...}
10 RSP 2.04
11 POST /oic/sec/acl2 {"aclist2": [...,{"subject":"uuid":"A71C3xxx-...", "permission":6, "resources":["rt":"x.a.b.c"]},...],...}
12 RSP 2.04
Enable ready-for-normal-operation device state.
13 POST /oic/sec/pstat[{"dos":{"s":3}}]
14 RSP 2.04
15 POST /oic/sec/pstat[{"dos":{"s":3}}]
16 RSP 2.04
CMS Device (UUID C85xxxxx-...)
AMS Device (UUID A85xxxxx-...)
New Device (UUID A71C3xxx-...)
Peer Device (UUID CA401xxx-...)

Figure 24 - Provision New Device for Peer to Peer Interaction Sequence

Step	Description
1 - 4	The OBT set the Devices in the ready for provisioning status by setting oic.sec.pstat.dos to 2.
5 - 8	The OBT provision the Device with peer credentials
9 - 12	The OBT provision the Device with access control entities for peer Devices.
13 - 16	Enable Device to RFNOP state by setting oic.sec.pstat.dos to 3.

Table 12 - Provision New Device for Peer to Peer Details

7.3.9 Security considerations regarding selecting an Ownership Transfer Method

An OBT and/or OBT's operator might have strict requirements for the list of OTMs that are acceptable when transferring ownership of a new Device. Some of the factors to be considered when determining those requirements are:

- The security considerations described above, for each of the OTMs
- The probability that a man-in-the-middle attacker might be present in the environment used to perform the Ownership Transfer

For example, the operator of an OBT might require that all of the Devices being onboarded support either the Random PIN or the Manufacturer Certificate OTM.

When such a local OTM policy exists, the OBT should try to use just the OTMs that are acceptable according to that policy, regardless of the doxm contents obtained during step 1 from the sequence diagram above (GET /oic/sec/doxm). If step 1 is performed over an unauthenticated and/or unencrypted connection between the OBT and the Device, the contents of the response to the GET request might have been tampered by a man-in-the-middle attacker. For example, the list of OTMs supported by the new Device might have been altered by the attacker.

Also, a man-in-the-middle attacker can force the DTLS session between the OBT and the new Device to fail. In such cases, the OBT has no way of determining if the session failed because the new Device doesn't support the OTM selected by the OBT, or because a man-in-the-middle injected such a failure into the communication between the OBT and the new Device.

The current version of this specification leaves the design and user experience related to the OTM policy mentioned above as OBT implementation details.

7.4 Provisioning

7.4.1 Provisioning Flows

As part of onboarding a new Device a secure channel is formed between the new Device and the OBT. Subsequent to the Device ownership status being changed to 'owned', there is an opportunity to begin provisioning. The OBT decides how the new Device will be managed going forward and provisions the support services that should be subsequently used to complete Device provisioning and on-going Device management.

The Device employs a Server-directed or Client-directed provisioning strategy. The /oic/sec/pstat Resource identifies the provisioning strategy and current provisioning status. The provisioning service should determine which provisioning strategy is most appropriate for the network. See Section 13.7 for additional detail.

7.4.1.1 Client-directed Provisioning

Client-directed provisioning relies on a provisioning service that identifies Servers in need of provisioning then performs all necessary provisioning duties.

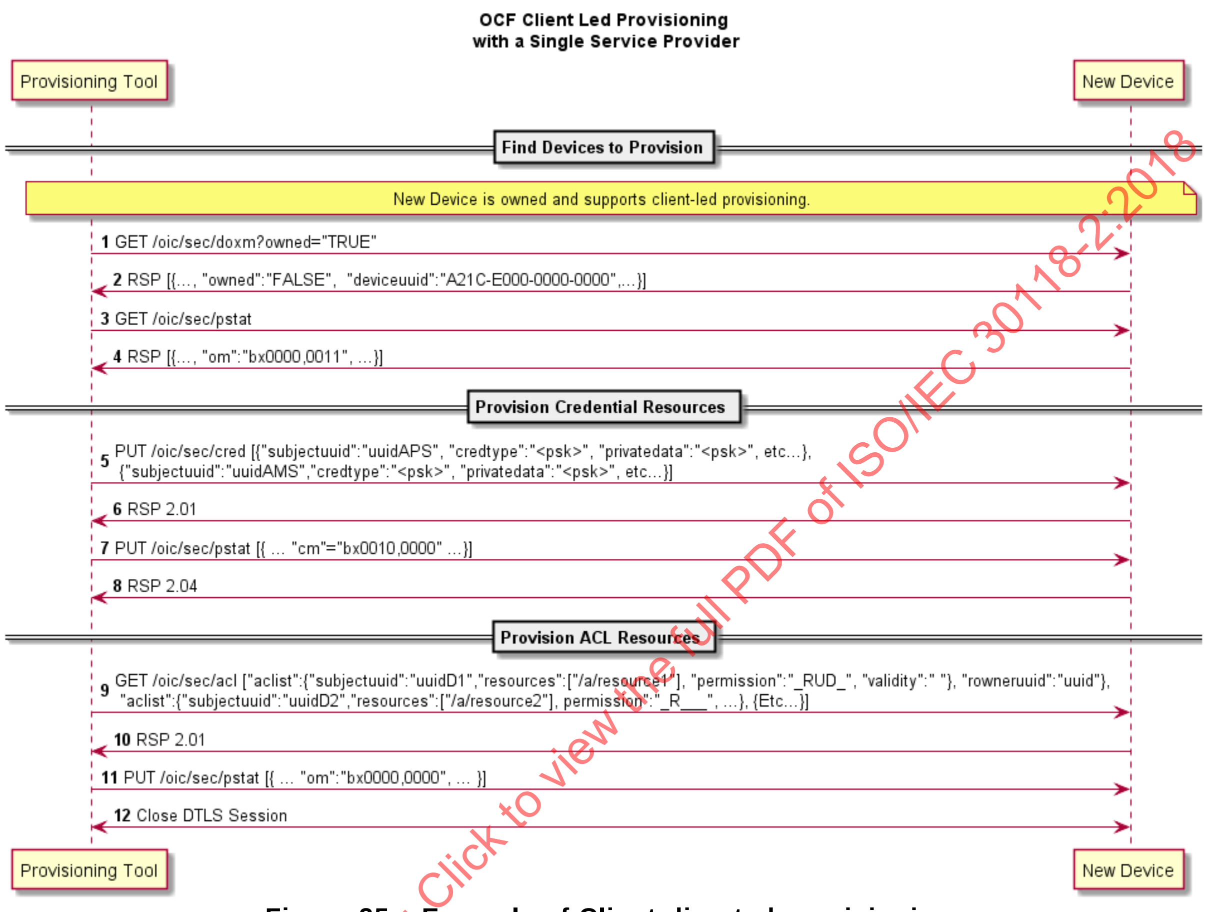

OCF Client Led Provisioning
with a Single Service Provider
Provisioning Tool
New Device
Find Devices to Provision
New Device is owned and supports client-led provisioning.
1 GET /oic/sec/doxm?owned="TRUE"
2 RSP [{..., "owned":"FALSE", "deviceuuid":"A21C-E000-0000-0000",...}]
3 GET /oic/sec/pstat
4 RSP [{..., "om":"bx0000,0011", ...}]
Provision Credential Resources
5 PUT /oic/sec/cred [{"subjectuuid":"uuidAPS", "credtype":"<psk>", "privatedata":"<psk>", etc...}, {"subjectuuid":"uuidAMS","credtype":"<psk>", "privatedata":"<psk>", etc...}]
6 RSP 2.01
7 PUT /oic/sec/pstat [{ ... "cm"="bx0010,0000" ...}]
8 RSP 2.04
Provision ACL Resources
9 GET /oic/sec/acl ["aclist":{"subjectuuid":"uuidD1","resources":["/a/resource1"], "permission":"_RUD_", "validity":" "}, "rowneruuid":"uuid"}, "aclist":{"subjectuuid":"uuidD2","resources":["/a/resource2"], permission":"_R___", ...}, {Etc...}]
10 RSP 2.01
11 PUT /oic/sec/pstat [{ ... "om":"bx0000,0000", ... }]
12 Close DTLS Session
Provisioning Tool
New Device

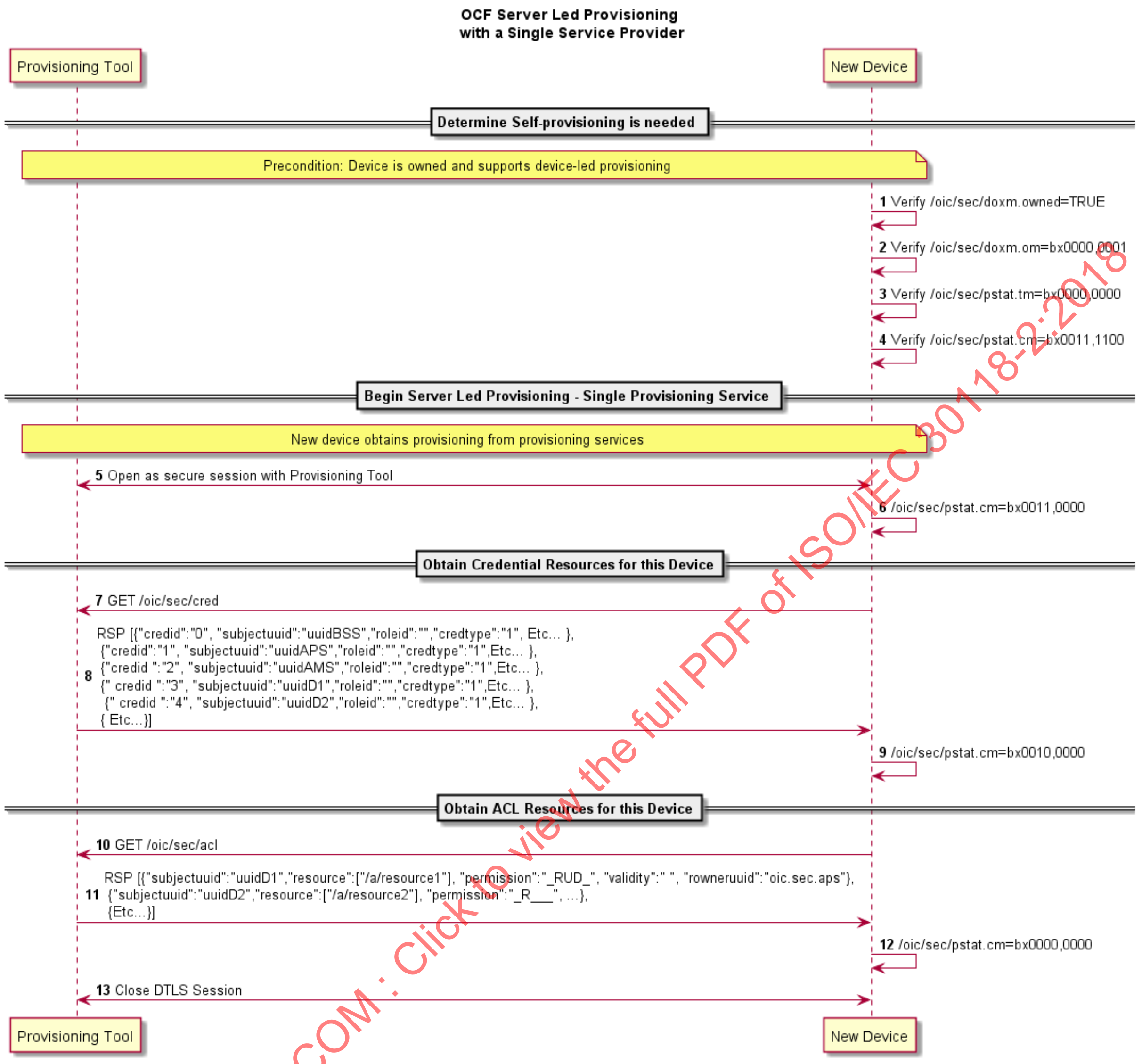
Figure 25 – Example of Client-directed provisioning

Step	Description
1	Discover Devices that are owned and support Client-directed provisioning.
2	The /oic/sec/doxm Resource identifies the Device and it's owned status.
3	PT obtains the new Device's provisioning status found in /oic/sec/pstat Resource
4	The pstat Resource describes the types of provisioning modes supported and which is currently configured. A Device manufacturer should set a default current operational mode (om). If the Om isn't configured for Client-directed provisioning, its om value can be changed.
5 - 6	Change state to Ready-for-Provisioning. cm is set to provision credentials and ACLs.
7 - 8	PT instantiates the /oic/sec/cred Resource. It contains credentials for the provisioned services and other Devices
9 - 10	cm is set to provision ACLs.
11 - 12	PT instantiates /oic/sec/acl Resources.
13 -14	The new Device provisioning status mode is updated to reflect that ACLs have been configured. (Ready-for-Normal-Operation state)
15	The secure session is closed.

Table 13 – Steps describing Client -directed provisioning

7.4.1.2 Server-directed Provisioning

Server-directed provisioning relies on the Server (i.e. New Device) for directing much of the provisioning work. As part of the onboarding process the support services used by the Server to seek additional provisioning are provisioned. The New Device uses a self-directed, state-driven approach to analyze current provisioning state, and tries to drive toward target state. This example assumes a single support service is used to provision the new Device.


OCF Server Led Provisioning
with a Single Service Provider
Provisioning Tool
New Device
Determine Self-provisioning is needed
Precondition: Device is owned and supports device-led provisioning
1 Verify /oic/sec/doxm.owned=TRUE
2 Verify /oic/sec/doxm.om=bx0000,0001
3 Verify /oic/sec/pstat.tm=bx0000,0000
4 Verify /oic/sec/pstat.cm=bx0011,1100
Begin Server Led Provisioning - Single Provisioning Service
New device obtains provisioning from provisioning services
5 Open as secure session with Provisioning Tool
6 /oic/sec/pstat.cm=bx0011,0000
Obtain Credential Resources for this Device
7 GET /oic/sec/cred
8
9 /oic/sec/pstat.cm=bx0010,0000
Obtain ACL Resources for this Device
10 GET /oic/sec/acl
11
12 /oic/sec/pstat.cm=bx0000,0000
13 Close DTLS Session
Provisioning Tool
New Device

```
RSP [{"credid":"0", "subjectuuid":"uuidBSS","roleid":"","credtype":"1", Etc... },
 {"credid":"1", "subjectuuid":"uuidAPS","roleid":"","credtype":"1",Etc... },
 {"credid ":"2", "subjectuuid":"uuidAMS","roleid":"","credtype":"1",Etc... },
 {" credid ":"3", "subjectuuid":"uuidD1","roleid":"","credtype":"1",Etc... },
  {" credid ":"4", "subjectuuid":"uuidD2","roleid":"","credtype":"1",Etc... },
 { Etc...}]
```

```
RSP [{"subjectuuid":"uuidD1","resource":["/a/resource1"], "permission":"_RUD_", "validity":" ", "rowneruuid":"oic.sec.aps"},
 {"subjectuuid":"uuidD2","resource":["/a/resource2"], "permission":"_R___", ...},
 {Etc...}]
```

Figure 26 – Example of Server-directed provisioning using a single provisioning service

Step	Description
1	The new Device verifies it is owned.
2	The new Device verifies it is in self-provisioning mode.
3	The new Device verifies its target provisioning state is fully provisioned.
4	The new Device verifies its current provisioning state requires provisioning.
5	The new Device initiates a secure session with the provisioning tool using the /oic/sec/doxm. DevOwner value to open a TLS connection using SharedKey.
7	The new Device updates Cm to reflect provisioning of bootstrap and other services.
8 – 9	The new Devices gets the /oic/sec/cred Resources. It contains credentials for the provisioned services and other Devices.
10	The new Device updates Cm to reflect provisioning of credential Resources.
11 – 12	The new Device gets the /oic/sec/acl Resources.
13	The new Device updates Cm to reflect provisioning of ACL Resources.
14	The secure session is closed.

Table 14 – Steps for Server-directed provisioning using a single provisioning service

7.4.1.3 Server-directed Provisioning Involving Multiple Support Services

A Server-directed provisioning flow, involving multiple support services distributes the provisioning work across multiple support services. Employing multiple support services is an effective way to distribute provisioning workload or to deploy specialized support. The following example demonstrates using a provisioning tool to configure two support services, a credential management support service and an ACL provisioning support service.

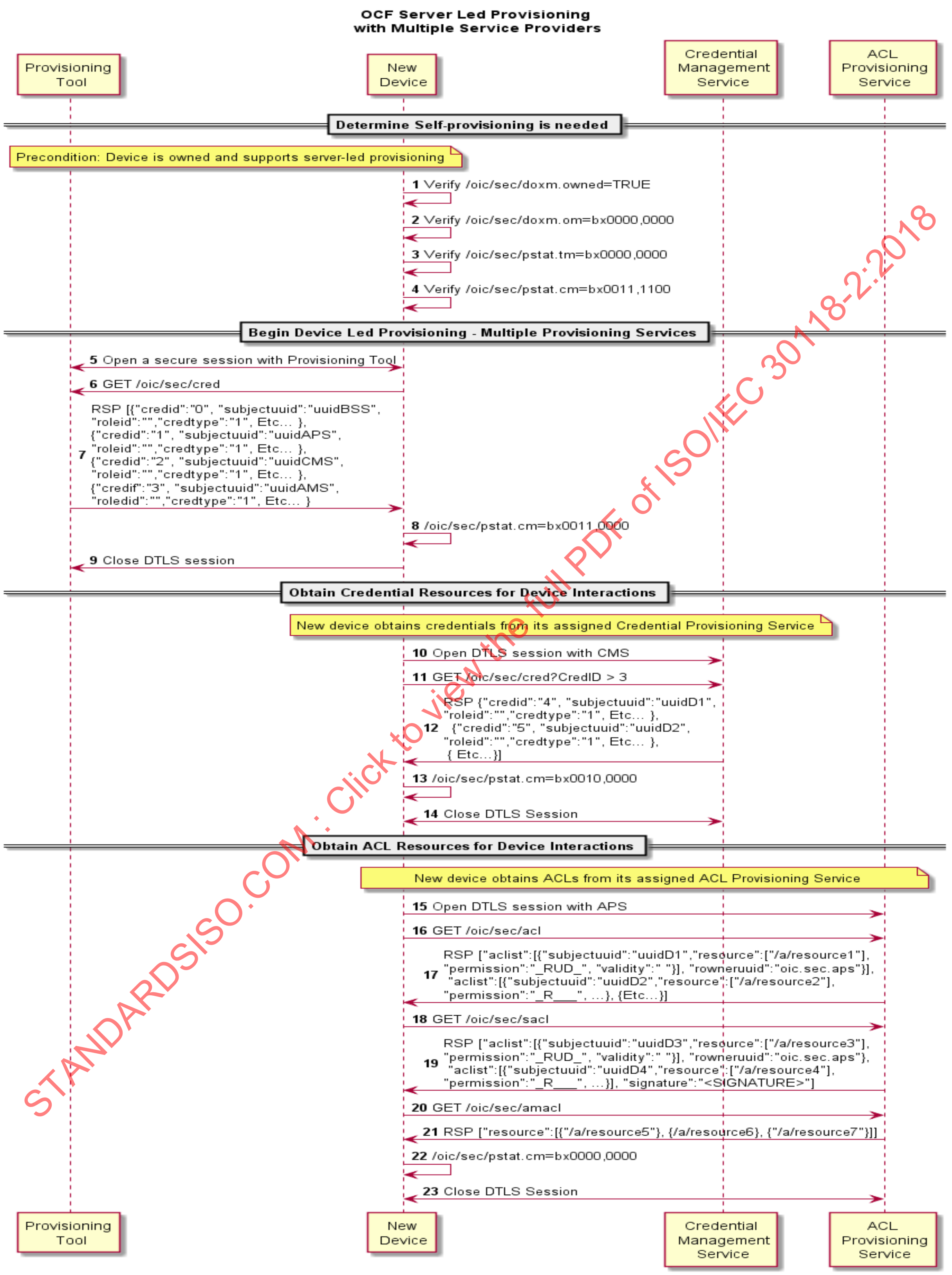

OCF Server Led Provisioning
with Multiple Service Providers
Provisioning Tool
New Device
Credential Management Service
ACL Provisioning Service
Determine Self-provisioning is needed
Precondition: Device is owned and supports server-led provisioning
1 Verify /oic/sec/doxm.owned=TRUE
2 Verify /oic/sec/doxm.om=bx0000,0000
3 Verify /oic/sec/pstat.tm=bx0000,0000
4 Verify /oic/sec/pstat.cm=bx0011,1100
Begin Device Led Provisioning - Multiple Provisioning Services
5 Open a secure session with Provisioning Tool
6 GET /oic/sec/cred
7 RSP [{"credid":"0", "subjectuuid":"uuidBSS", "roleid":"","credtype":"1", Etc... }, {"credid":"1", "subjectuuid":"uuidAPS", "roleid":"","credtype":"1", Etc... }, {"credid":"2", "subjectuuid":"uuidCMS", "roleid":"","credtype":"1", Etc... }, {"credif":"3", "subjectuuid":"uuidAMS", "roledid":"","credtype":"1", Etc... }
8 /oic/sec/pstat.cm=bx0011,0000
9 Close DTLS session
Obtain Credential Resources for Device Interactions
New device obtains credentials from its assigned Credential Provisioning Service
10 Open DTLS session with CMS
11 GET /oic/sec/cred?CredID > 3
12 RSP {"credid":"4", "subjectuuid":"uuidD1", "roleid":"","credtype":"1", Etc... }, {"credid":"5", "subjectuuid":"uuidD2", "roleid":"","credtype":"1", Etc... }, { Etc...}]
13 /oic/sec/pstat.cm=bx0010,0000
14 Close DTLS Session
Obtain ACL Resources for Device Interactions
New device obtains ACLs from its assigned ACL Provisioning Service
15 Open DTLS session with APS
16 GET /oic/sec/acl
17 RSP ["aclist":[{"subjectuuid":"uuidD1","resource":["/a/resource1"], "permission":"_RUD_", "validity":" "}], "rowneruuid":"oic.sec.aps"}], "aclist":[{"subjectuuid":"uuidD2","resource":["/a/resource2"], "permission":"_R___", ...}, {Etc...}]
18 GET /oic/sec/sacl
19 RSP ["aclist":[{"subjectuuid":"uuidD3","resource":["/a/resource3"], "permission":"_RUD_", "validity":" "}], "rowneruuid":"oic.sec.aps"}, "aclist":[{"subjectuuid":"uuidD4","resource":["/a/resource4"], "permission":"_R___", ...}], "signature":"<SIGNATURE>"]
20 GET /oic/sec/amacl
21 RSP ["resource":[{"/a/resource5"}, {/a/resource6}, {"/a/resource7"}]]
22 /oic/sec/pstat.cm=bx0000,0000
23 Close DTLS Session
Provisioning Tool
New Device
Credential Management Service
ACL Provisioning Service

Figure 27 – Example of Server-directed provisioning involving multiple support services

Step	Description
1	The new Device verifies it is owned.
2	The new Device verifies it is in self-provisioning mode.
3	The new Device verifies its target provisioning state is fully provisioned.
4	The new Device verifies its current provisioning state requires provisioning.
5	The new Device initiates a secure session with the provisioning tool using the /oic/sec/doxm. DevOwner value to open a TLS connection using SharedKey.
6	The new Device updates Cm to reflect provisioning of support services.
7	The new Device closes the DTLS session with the provisioning tool.
8	The new Device finds the CMS from the /oic/sec/cred Resource, rowneruuid property and opens a DTLS connection. The new device finds the credential to use from the /oic/sec/cred Resource.
9 – 10	The new Device requests additional credentials that are needed for interaction with other devices.
11	The new Device updates Cm to reflect provisioning of credential Resources.
12	The DTLS connection is closed.
13	The new Device finds the ACL provisioning and management service from the /oic/sec/acl2 Resource, rowneruuid property and opens a DTLS connection. The new device finds the credential to use from the /oic/sec/cred Resource.
14 – 15	The new Device gets ACL Resources that it will use to enforce access to local Resources.
16 – 18	The new Device should get SACL Resources immediately or in response to a subsequent Device Resource request.
19 – 20	The new Device should also get a list of Resources that should consult an Access Manager for making the access control decision.
21	The new Device updates Cm to reflect provisioning of ACL Resources.
22	The DTLS connection is closed.

Table 15 – Steps for Server-directed provisioning involving multiple support services

7.5 Bootstrap Example

This section is left intentionally blank.

8 Device Onboarding State Definitions

As explained in Section 5.2, the process of onboarding completes after the ownership of the Device has been transferred and the Device has been provisioned with relevant configuration/services as explained in Section 5.3. The diagram below shows the various states a Device can be in during the Device lifecycle.

The /pstat.dos.s property is RW by the /pstat resource owner (e.g. 'doxs' or 'bss' service) so that the resource owner can remotely update the Device state. When the Device is in RFNOP or RFPRO, ACLs can be used to allow remote control of Device state by other Devices. When the Device state is SRESET the Device owner credential may be the only indication of authorization to access the Device. The Device owner may perform low-level consistency checks and re-provisioning to get the Device suitable for a transition to RFPRO.

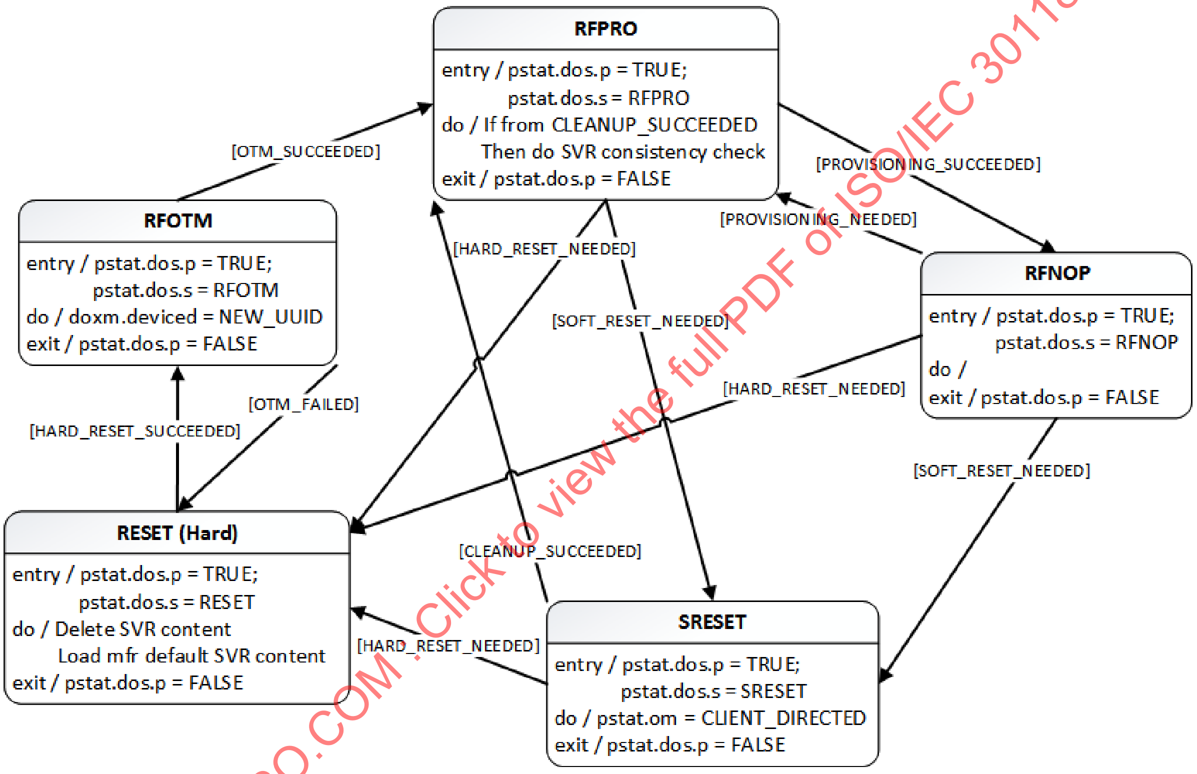

RFPRO
entry / pstat.dos.p = TRUE;
pstat.dos.s = RFPRO
do / If from CLEANUP_SUCCEEDED
Then do SVR consistency check
exit / pstat.dos.p = FALSE
RFOTM
entry / pstat.dos.p = TRUE;
pstat.dos.s = RFOTM
do / doxm.deviced = NEW_UUID
exit / pstat.dos.p = FALSE
RFNOP
entry / pstat.dos.p = TRUE;
pstat.dos.s = RFNOP
do /
exit / pstat.dos.p = FALSE
RESET (Hard)
entry / pstat.dos.p = TRUE;
pstat.dos.s = RESET
do / Delete SVR content
Load mfr default SVR content
exit / pstat.dos.p = FALSE
SRESET
entry / pstat.dos.p = TRUE;
pstat.dos.s = SRESET
do / pstat.om = CLIENT_DIRECTED
exit / pstat.dos.p = FALSE
[OTM_SUCCEEDED]
[PROVISIONING_SUCCEEDED]
[PROVISIONING_NEEDED]
[HARD_RESET_NEEDED]
[SOFT_RESET_NEEDED]
[OTM_FAILED]
[HARD_RESET_SUCCEEDED]
[HARD_RESET_NEEDED]
[SOFT_RESET_NEEDED]
[CLEANUP_SUCCEEDED]
[HARD_RESET_NEEDED]

Figure 28 – Device state model

As shown in the diagram, at the conclusion of the provisioning step, the Device comes in the "Ready for Normal Operation" state where it has all it needs in order to start interoperating with other Devices. Section 8.1 specifies the minimum mandatory configuration that a Device shall hold in order to be considered as "Ready for Normal Operation".

In the event of power loss or Device failure, the Device should remain in the same state that it was in prior to the power loss / failure

If a Device or resource owner OBSERVEs /pstat.dos.s, then transitions to SRESET will give early warning notification of Devices that may require SVR consistency checking.

In order for onboarding to function, the Device shall have the following Resources installed:

1. /oic/sec/doxm Resource

2. /oic/sec/pstat Resource
3. /oic/sec/cred Resource

The values contained in these Resources are specified in the state definitions below.

8.1 Device Onboarding-Reset State Definition

The /pstat.dos.s = RESET state is defined as a "hard" reset to manufacturer defaults. Hard reset also defines a state where the Device asset is ready to be transferred to another party.

The Platform manufacturer should provide a physical mechanism (e.g. button) that forces Platform reset. All Devices hosted on the same Platform transition their Device states to RESET when the Platform reset is asserted.

The following Resources and their specific properties shall have the value as specified.

1. The "owned" Property of the /oic/sec/doxm Resource shall transition to FALSE.
2. The "devowneruuid" Property of the /oic/sec/doxm Resource shall be nil UUID.
3. The "devowner" Property of the /oic/sec/doxm Resource shall be nil UUID, if this Property is implemented.
4. The "deviceuuid" Property of the /oic/sec/doxm Resource shall be reset to the manufacturer’s default value.
5. The "deviceid" Property of the /oic/sec/doxm Resource shall be reset to the manufacturer’s default value, if this Property is implemented.
6. The "sct" Property of the /oic/sec/doxm Resource shall be reset to the manufacturer’s default value.
7. The "oxmsel" Property of the /oic/sec/doxm Resource shall be reset to the manufacturer’s default value.
8. The "isop" Property of the /oic/sec/pstat Resource shall be FALSE.
9. The "dos" of the /oic/sec/pstat Resource shall be updated: dos.s shall equal "RESET" state and dos.p shall equal "FALSE".
10. The current provisioning mode Property - "cm" of the /oic/sec/pstat Resource shall be "00000001".
11. The target provisioning mode Property - "tm" of the /oic/sec/pstat Resource shall be "00000010".
12. The operational modes Property - "om" of the /oic/sec/pstat Resource shall be set to the manufacturer default value.
13. The supported operational modes Property - "sm" of the /oic/sec/pstat Resource shall be set to the manufacturer default value.
14. The "rowneruuid" Property of /oic/sec/pstat, /oic/sec/doxm, /oic/sec/acl, /oic/sec/amacl, /oic/sec/sacl, and /oic/sec/cred Resources shall be nil UUID.

.

8.2 Device Ready-for-OTM State Definition

The following Resources and their specific properties shall have the value as specified for an operational Device that is ready for ownership transfer

1. The "owned" Property of the /oic/sec/doxm Resource shall be FALSE and will transition to TRUE.

2. The "devowner" Property of the /oic/sec/doxm Resource shall be nil UUID, if this Property is implemented.

3. The "devowneruuid" Property of the /oic/sec/doxm Resource shall be nil UUID.

4. The "deviceid" Property of the /oic/sec/doxm Resource may be nil UUID, if this Property is implemented. The value of the "di" Property in /oic/d is undefined.

5. The "deviceuuid" Property of the /oic/sec/doxm Resource may be nil UUID. The value of the "di" Property in /oic/d is undefined.

6. The "isop" Property of the /oic/sec/pstat Resource shall be FALSE.

7. The "dos" of the /oic/sec/pstat Resource shall be updated: dos.s shall equal "RFOTM" state and dos.p shall equal "FALSE".

8. The "cm" Property of the /oic/sec/pstat Resource shall be "00XXXX10".

9. The "tm" Property of the /oic/sec/pstat shall be "00XXXX00".

10. The /oic/sec/cred Resource should contain credential(s) if required by the selected OTM

8.3 Device Ready-for-Provisioning State Definition

The following Resources and their specific properties shall have the value as specified when the Device is ready for additional provisioning:

1. The "owned" Property of the /oic/sec/doxm Resource shall be TRUE.

2. The "devowneruuid" Property of the /oic/sec/doxm Resource shall not be nil UUID.

3. The "deviceuuid" Property of the /oic/sec/doxm Resource shall not be nil UUID and shall be set to the value that was determined during RFOTM processing. Also the value of the "di" Property in /oic/d Resource shall be the same as the deviceid Property in the /oic/sec/doxm Resource.

4. The "oxmsel" Property of the /oic/sec/doxm Resource shall have the value of the actual OTM used during ownership transfer.

5. The "isop" Property of the /oic/sec/pstat Resource shall be FALSE.

6. The "dos" of the /oic/sec/pstat Resource shall be updated: dos.s shall equal "RFPRO" state and dos.p shall equal "FALSE".

7. The "cm" Property of the /oic/sec/pstat Resource shall be "00XXXX00".

8. The "tm" Property of the /oic/sec/pstat shall be "00XXXX00".

9. The "rowneruuid" Property of every installed Resource shall be set to a valid Resource owner (i.e. an entity that is authorized to instantiate or update the given Resource). Failure to set a rowneruuid or rowner (at least one of the two) may result in an orphan Resource.

10. The /oic/sec/cred Resource shall contain credentials for each entity referenced by an rowneruuid, amsuuid, devowneruuid.

8.4 Device Ready-for-Normal-Operation State Definition

The following Resources and their specific properties shall have the value as specified for an operational Device Final State

1. The "owned" Property of the /oic/sec/doxm Resource shall be TRUE.
2. The "devowneruuid" Property of the /oic/sec/doxm Resource shall not be nil UUID.
3. The "deviceuuid" Property of the /oic/sec/doxm Resource shall not be nil UUID and shall be set to the ID that was configured during OTM. Also the value of the "di" Property in /oic/d shall be the same as the deviceuuid.
4. The "oxmsel" Property of the /oic/sec/doxm Resource shall have the value of the actual OTM used during ownership transfer.
5. The "isop" Property of the /oic/sec/pstat Resource shall be TRUE.
6. The "dos" of the /oic/sec/pstat Resource shall be updated: dos.s shall equal "RFNOP" state and dos.p shall equal "FALSE".
7. The "cm" Property of the /oic/sec/pstat Resource shall be "00XXXX00" (where "X" is interpreted as either 1 or 0).
8. The "tm" Property of the /oic/sec/pstat shall be "00XXXX00".
9. The "rowneruuid" Property of every installed Resource shall be set to a valid resource owner (i.e. an entity that is authorized to instantiate or update the given Resource). Failure to set a rowneruuid or rowner (at least one of the two) may result in an orphan Resource.
10. The /oic/sec/cred Resource shall contain credentials for each service referenced by a rowneruuid, amsuuid, devowneruuid.

8.5 Device Soft Reset State Definition

The soft reset state is defined (e.g. /pstat.dos.s = SRESET) where entrance into this state means the Device is not operational but remains owned by the current owner. The Device may exit SRESET by authenticating to an OBT (e.g. "rt" = "oic.r.doxs") using the OC provided during original onboarding (but should not require use of an owner transfer method /doxm.oxms).

The OBT should perform a consistency check of the SVR and if necessary, re-provision them sufficiently to allow the Device to transition to RFPRO.

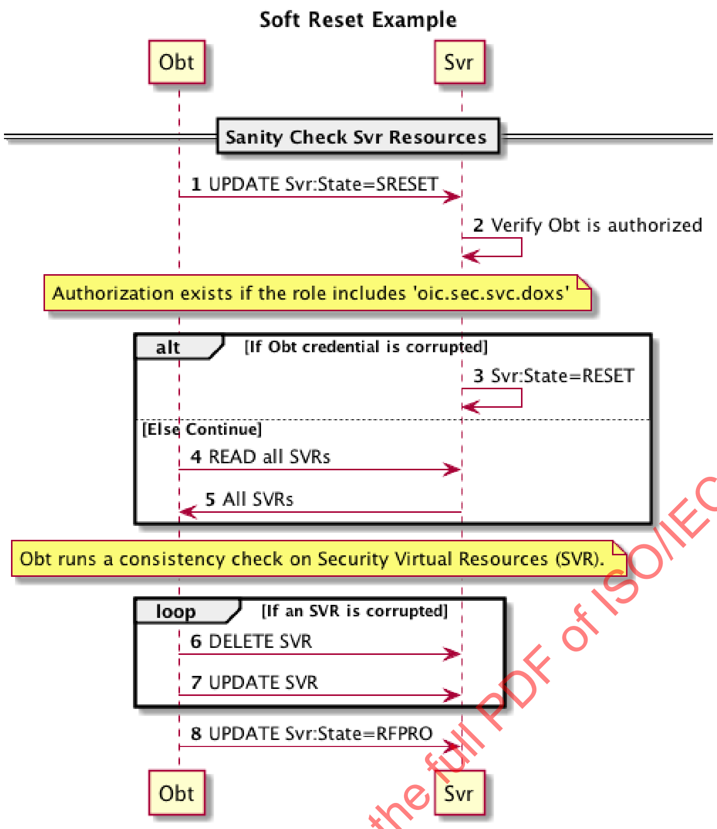

Soft Reset Example
Obt
Svr
Sanity Check Svr Resources
1 UPDATE Svr:State=SRESET
2 Verify Obt is authorized
Authorization exists if the role includes 'oic.sec.svc.doxs'
alt
[If Obt credential is corrupted]
3 Svr:State=RESET
[Else Continue]
4 READ all SVRs
5 All SVRs
Obt runs a consistency check on Security Virtual Resources (SVR).
loop
[If an SVR is corrupted]
6 DELETE SVR
7 UPDATE SVR
8 UPDATE Svr:State=RFPRO
Obt
Svr

Figure 29 – OBT Sanity Check Sequence in SRESET

The OBT should perform a sanity check of SVRs before final transition to RFPRO Device state. If the Device's OBT credential cannot be found or is determined to be corrupted, the Device state transitions to RESET. The Device should remain in SRESET if the OBT credential fails to validate the OBT. This mitigates denial-of-service attacks that may be attempted by non-OBT Devices.

When in SRESET, the following Resources and their specific Properties shall have the values as specified.

1. The "owned" Property of the /oic/sec/doxm Resource shall be TRUE.
2. The "devowneruuid" Property of the /oic/sec/doxm Resource shall remain non-null.
3. The "devowner" Property of the /oic/sec/doxm Resource shall be non-null, if this Property is implemented.
4. The "deviceuuid" Property of the /oic/sec/doxm Resource shall remain non-null.
5. The "deviceid" Property of the /oic/sec/doxm Resource shall remain non-null.
6. The "sct" Property of the /oic/sec/doxm Resource shall retain its value.
7. The "oxmsel" Property of the /oic/sec/doxm Resource shall retains its value.
8. The "isop" Property of the /oic/sec/pstat Resource shall be FALSE.
9. The /oic/sec/pstat.dos.s Property shall be SRESET.

10. The current provisioning mode Property - "cm" of the /oic/sec/pstat Resource shall be "00000001".

11. The target provisioning mode Property - "tm" of the /oic/sec/pstat Resource shall be "00XXXX00".

12. The operational modes Property - "om" of the /oic/sec/pstat Resource shall be ‘client-directed mode’.

13. The supported operational modes Property (/pstat.sm) may be updated by the Device owner (aka DOXS).

14. The "rowneruuid" Property of /oic/sec/pstat, /oic/sec/doxm, /oic/sec/acl, /oic/sec/acl2, /oic/sec/amacl, /oic/sec/sacl, and /oic/sec/cred Resources may be reset by the Device owner (aka DOXS) and re-provisioned.

9 Security Credential Management

This section provides an overview of the credential types in OCF, along with details of credential use, provisioning and ongoing management.

9.1 Credential Lifecycle

OCF credential lifecycle has the following phases: (1) creation, (2) deletion, (3) refresh, (4) issuance and (5) revocation.

9.1.1 Creation

Devices may instantiate credential Resources directly using an ad-hoc key exchange method such as Diffie-Hellman. Alternatively, a CMS may be used to provision credential Resources to the Device.

The credential Resource maintains a resource owner Property (/oic/sec/cred.Rowner) that identifies a CMS. If a credential was created ad-hoc, the peer Device involved in the Key Exchange is considered to be the CMS.

Credential Resources created using a CMS may involve specialized credential issuance protocols and messages. These may involve the use of public key infrastructure (PKI) such as a certificate authority (CA), symmetric key management such as a key distribution centre (KDC) or as part of a provisioning action by a provisioning, bootstrap or onboarding service.

9.1.2 Deletion

The CMS can delete credential Resources or the Device (e.g. the Device where the credential Resource is hosted) can directly delete credential Resources.

An expired credential Resource may be deleted to manage memory and storage space.

Deletion in OCF key management is equivalent to credential suspension.

9.1.3 Refresh

Credential refresh may be performed with the help of a CMS before it expires.

The method used to obtain the credential initially should be used to refresh the credential.

The /oic/sec/cred Resource supports expiry using the Period Property. Credential refresh may be applied when a credential is about to expire or is about to exceed a maximum threshold for bytes encrypted.

A credential refresh method specifies the options available when performing key refresh. The Period Property informs when the credential should expire. The Device may proactively obtain a new credential using a credential refresh method using current unexpired credentials to refresh the existing credential. If the Device does not have an internal time source, the current time should be obtained from a CMS at regular intervals.

Alternatively, a CMS can be used to refresh or re-issue an expired credential unless no trusted CMS can be found.

If the CMS credential is allowed to expire, the BSS or onboarding service may be used to re-provision the CMS. If the onboarding established credentials are allowed to expire the Device will need to be re-onboarded and the device owner transfer steps re-applied.

If credentials established through ad-hoc methods are allowed to expire the ad-hoc methods will need to be re-applied.

All Devices shall support at least one credential refresh method.

9.1.4 Revocation

Credentials issued by a CMS may be equipped with revocation capabilities. In situations where the revocation method involves provisioning of a revocation object that identifies a credential that is to be revoked prior to its normal expiration period, a credential Resource is created containing the revocation information that supersedes the originally issued credential. The revocation object expiration should match that of the revoked credential so that the revocation object is cleaned up upon expiry.

It is conceptually reasonable to consider revocation applying to a credential or to a Device. Device revocation asserts all credentials associated with the revoked Device should be considered for revocation. Device revocation is necessary when a Device is lost, stolen or compromised. Deletion of credentials on a revoked Device might not be possible or reliable.

9.2 Credential Types

The /oic/sec/cred Resource maintains a credential type Property that supports several cryptographic keys and other information used for authentication and data protection. The credential types supported include pair-wise symmetric keys, group symmetric keys, asymmetric authentication keys, certificates (i.e. signed asymmetric keys) and shared-secrets (i.e. PIN/password).

9.2.1 Pair-wise Symmetric Key Credentials

Pair-wise symmetric key credentials have a symmetric key in common with exactly one other peer Device. A CMS might maintain an instance of the symmetric key. The CMS is trusted to issue or provision pair-wise keys and not misuse it to masquerade as one of the pair-wise peers.

Pair-wise keys could be established through ad-hoc key agreement protocols.

The PrivateData Property in the /oic/sec/cred Resource contains the symmetric key.

The PublicData Property may contain a token encrypted to the peer Device containing the pair-wise key.

The OptionalData Property may contain revocation status.

The Device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The Device implementer should apply appropriate integrity, confidentiality and access protection of the /oic/sec/cred, /oic/sec/crl, /oic/sec/roles, /oic/sec/csr Resources to prevent unauthorized modifications.

9.2.2 Group Symmetric Key Credentials

Group keys are symmetric keys shared among a group of Devices (3 or more). Group keys are used for efficient sharing of data among group participants.

Group keys do not provide authentication of Devices but only establish membership in a group.

Group keys are distributed with the aid of a CMS. The CMS is trusted to issue or provision group keys and not misuse them to manipulate protected data.

The PrivateData Property in the /oic/sec/cred Resource contains the symmetric key.

The PublicData Property may contain the group name.

The OptionalData Property may contain revocation status.

The Device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The Device implementer should apply appropriate integrity, confidentiality and access protection of the /oic/sec/cred, /oic/sec/crl, /oic/sec/roles, /oic/sec/csr Resources to prevent unauthorized modifications.

9.2.3 Asymmetric Authentication Key Credentials

Asymmetric authentication key credentials contain either a public and private key pair or only a public key. The private key is used to sign Device authentication challenges. The public key is used to verify a device authentication challenge-response.

The PrivateData Property in the /oic/sec/cred Resource contains the private key.

The PublicData Property contains the public key.

The OptionalData Property may contain revocation status.

The Device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

Devices should generate asymmetric authentication key pairs internally to ensure the private key is only known by the Device. See Section 9.2.3.1 for when it is necessary to transport private key material between Devices.

The Device implementer should apply appropriate integrity, confidentiality and access protection of the /oic/sec/cred, /oic/sec/crl, /oic/sec/roles, /oic/sec/csr Resources to prevent unauthorized modifications.

9.2.3.1 External Creation of Asymmetric Authentication Key Credentials

Devices should employ industry-standard high-assurance techniques when allowing off-device key pair creation and provisioning. Use of such key pairs should be minimized, particularly if the key pair is immutable and cannot be changed or replaced after provisioning.

When used as part of onboarding, these key pairs can be used to prove the Device possesses the manufacturer-asserted properties in a certificate to convince an OBT or a user to accept onboarding the Device. See Section 7.3.3 for the owner transfer method that uses such a certificate to authenticate the Device, and then provisions new network credentials for use.

9.2.4 Asymmetric Key Encryption Key Credentials

The asymmetric key-encryption-key (KEK) credentials are used to wrap symmetric keys when distributing or storing the key.

The PrivateData Property in the /oic/sec/cred Resource contains the private key.

The PublicData Property contains the public key.

The OptionalData Property may contain revocation status.

The Device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The Device implementer should apply appropriate integrity, confidentiality and access protection of the /oic/sec/cred, /oic/sec/crl, /oic/sec/roles, /oic/sec/csr Resources to prevent unauthorized modifications.

9.2.5 Certificate Credentials

Certificate credentials are asymmetric keys that are accompanied by a certificate issued by a CMS or an external certificate authority (CA).

A certificate enrolment protocol is used to obtain a certificate and establish proof-of-possession.

The issued certificate is stored with the asymmetric key credential Resource.

Other objects useful in managing certificate lifecycle such as certificate revocation status are associated with the credential Resource.

Either an asymmetric key credential Resource or a self-signed certificate credential is used to terminate a path validation.

The PrivateData Property in the /oic/sec/cred Resource contains the private key.

The PublicData Property contains the issued certificate.

The OptionalData Property may contain revocation status.

The Device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The Device implementer should apply appropriate integrity, confidentiality and access protection of the /oic/sec/cred, /oic/sec/crl, /oic/sec/roles, /oic/sec/csr Resources to prevent unauthorized modifications.

9.2.6 Password Credentials

Shared secret credentials are used to maintain a PIN or password that authorizes Device access to a foreign system or Device that doesn't support any other OCF credential types.

The PrivateData Property in the /oic/sec/cred Resource contains the PIN, password and other values useful for changing and verifying the password.

The PublicData Property may contain the user or account name if applicable.

The OptionalData Property may contain revocation status.

The Device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The Device implementer should apply appropriate integrity, confidentiality and access protection of the /oic/sec/cred, /oic/sec/crl, /oic/sec/roles, /oic/sec/csr Resources to prevent unauthorized modifications.

9.3 Certificate Based Key Management

9.3.1 Overview

To achieve authentication and transport security during communications in OCF network, certificates containing public keys of communicating parties and private keys can be used.

The certificate and private key may be issued by a local or remote certificate authority (CA) when a Device is deployed in the OCF network and credential provisioning is supported by a CMS (Credential Management Service). For the local CA, a certificate revocation list (CRL) based on X.509 is used to validate proof of identity. In the case of a remote CA, Online Certificate Status Protocol (OCSP) can be used to validate proof of identity and validity.

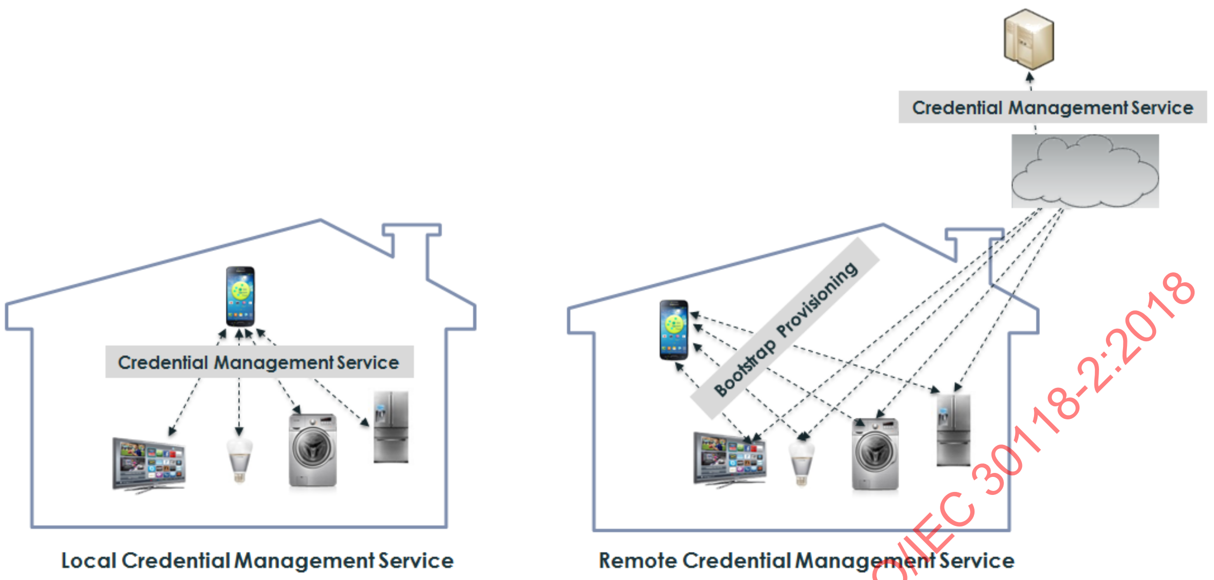

Credential Management Service
Credential Management Service
Bootstrap Provisioning
Local Credential Management Service
Remote Credential Management Service

Figure 30 – Certificate Management Architecture

The OCF certificate and OCF CRL (Certificate Revocation List) format is a subset of X.509 format, only elliptic curve algorithm and DER encoding format are allowed, most of optional fields in X.509 are not supported so that the format intends to meet the constrained Device's requirement.

As for the certificate and CRL management in the Server, the process of storing, retrieving and parsing Resources of the certificates and CRL will be performed at the security resource manager layer; the relevant Interfaces may be exposed to the upper layer.

A SRM is the security enforcement point in a Server as described in Section 5.4, so the data of certificates and CRL will be stored and managed in SVR database.

The request to issue a Device's certificate should be managed by a CMS when a Device is newly onboarded or the certificate of the Device is revoked. When a certificate is considered invalid, it must be revoked. A CRL is a data structure containing the list of revoked certificates and their corresponding Devices that are not be trusted. The CRL is expected to be regularly updated (for example; every 3 months) in real operations.

9.3.2 Certificate Format

An OCF certificate format is a subset of X.509 format (version 3 or above) as defined in [RFC5280].

9.3.2.1 Certificate Profile and Fields

The OCF certificate shall support the following fields; `version`, `serialNumber`, `signature`, `issuer`, `validity`, `subject`, `subjectPublicKeyInfo`, `extensions`, `signatureAlgorithm` and `signatureValue`.

- `version`: the version of the encoded certificate
- `serialNumber` : certificate serial number
- `signature`: the algorithm identifier for the algorithm used by the CA to sign this certificate
- `issuer`: the entity that has signed and issued certificates
- `validity`: the time interval during which CA warrants

- `subject`: the entity associated with the subject public key field (deviceID)
- `subjectPublicKeyInfo`: the public key and the algorithm with which key is used
- `extensions`: certificate extensions as defined in section 9.3.2.2
- `signatureAlgorithm`: the cryptographic algorithm used by the CA to sign this certificate
- `signatureValue`: the digital signature computed upon the ASN.1 DER encoded `OCFtbsCertificate` (this signature value is encoded as a BIT STRING.)

The OCF certificate syntax shall be defined as follows;

```
OCFCertificate  ::=  SEQUENCE  {

        OCFtbsCertificate       TBSCertificate,
        signatureAlgorithm      AlgorithmIdentifier,
        signatureValue          BIT STRING
}
```

The `OCFtbsCertificate` field contains the names of a subject and an issuer, a public key associated with the subject, a validity period, and other associated information. Per RFC5280, version 3 certificates use the value 2 in the version field to encode the version number; the below grammar does not allow version 2 certificates.

```
OCFtbsCertificate  ::=  SEQUENCE  {
        version             [0]  2 or above,
        serialNumber        CertificateSerialNumber,
        signature           AlgorithmIdentifier,
        issuer              Name,
        validity            Validity,
        subject             Name,
    subjectPublicKeyInfo  SubjectPublicKeyInfo,
        extensions      [3] EXPLICIT Extensions
}
subjectPublicKeyInfo  ::=  SEQUENCE  {
        algorithm              AlgorithmIdentifier,
        subjectPublicKey       BIT STRING
}
Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
       extnID        OBJECT IDENTIFIER,
       critical      BOOLEAN DEFAULT FALSE,
       extnValue     OCTET STRING
                     -- contains the DER encoding of an ASN.1 value
                     -- corresponding to the extension type identified
                     -- by extnID
}
```

Certificate Fields		Description	OCF	X.509
`OCFtbsCertificate`	`version`	2 or above	Mandatory	Mandatory
	`serialNumber`	CertificateSerialNumber	Mandatory	Mandatory
	`signature`	AlgorithmIdentifier	1.2.840.10045.4.3.2(ECDSA algorithm with SHA256,	Specified in [RFC3279],[RFC

			Mandatory)	4055], and [RFC4491]
	issuer	Name	Mandatory	Mandatory
	validity	Validity	Mandatory	Mandatory
	subject	Name	Mandatory	Mandatory
	subjectPublicKeyInfo	SubjectPublicKeyInfo	1.2.840.10045.2.1, 1.2.840.10045.3.1.7(ECDSA algorithm with SHA256 based on secp256r1 curve, Mandatory)	Specified in [RFC3279],[RFC 4055], and [RFC4491]
	issuerUniqueID	IMPLICIT UniqueIdentifier	Not supported	Optional
	subjectUniqueID	IMPLICIT UniqueIdentifier	Not supported	
	extensions	EXPLICIT Extensions	Mandatory	
signatureAlgorithm		AlgorithmIdentifier	1.2.840.10045.4.3.2(ECDSA algorithm with SHA256, Mandatory)	Specified in [RFC3279],[RFC 4055], and [RFC4491]
signatureValue		BIT STRING	Mandatory	Mandatory

Table 16 – Comparison between OCF and X.509 certificate fields

9.3.2.2 Supported Certificate Extensions

As these certificate extensions are a standard part of RFC 5280, this specification includes the section number from that RFC to include it by reference. Each extension is summarized here, and any modifications to the RFC definition are listed. Devices MUST implement and understand the extensions listed here; other extensions from the RFC are not included in this specification and therefore are not required. Section 10.3 describes what Devices must implement when validating certificate chains, including processing of extensions, and actions to take when certain extensions are absent.

- Authority Key Identifier (4.2.1.1)

  The Authority Key Identifier (AKI) extension provides a means of identifying the public key corresponding to the private key used to sign a certificate. This specification makes the following modifications to the referenced definition of this extension:

  The authorityCertIssuer or authorityCertSerialNumber fields of the AuthorityKeyIdentifier sequence are not permitted; only keyIdentifier is allowed. This results in the following grammar definition:

```
id-ce-authorityKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 35 }

AuthorityKeyIdentifier ::= SEQUENCE {
      keyIdentifier             [0] KeyIdentifier           }

KeyIdentifier ::= OCTET STRING
```

- Subject Key Identifier (4.2.1.2)

  The Subject Key Identifier (SKI) extension provides a means of identifying certificates that contain a particular public key.

This specification makes the following modification to the referenced definition of this extension:

Subject Key Identifiers SHOULD be derived from the public key contained in the certificate's SubjectPublicKeyInfo field or a method that generates unique values. This specification RECOMMENDS the 256-bit SHA-2 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits). Devices verifying certificate chains must not assume any particular method of computing key identifiers, however, and must only base matching AKI's and SKI's in certification path constructions on key identifiers seen in certificates.

- Subject Alternative Name

If the EKU extension is present, and has the value XXXXXX, indicating that this is a role certificate, the Subject Alternative Name (subjectAltName) extension shall be present and interpreted as described below. When no EKU is present, or has another value, the subjectAltName extension SHOULD be absent. The subjectAltName extension is used to encode one or more Role ID values in role certificates, binding the roles to the subject public key. The subjectAltName extension is defined in RFC 5280 (Section 4.2.1.6):

```
id-ce-subjectAltName OBJECT IDENTIFIER ::=  { id-ce 17 }

SubjectAltName ::= GeneralNames

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
        otherName                       [0]     OtherName,
        rfc822Name                      [1]     IA5String,
        dNSName                         [2]     IA5String,
        x400Address                     [3]     ORAddress,
        directoryName                   [4]     Name,
        ediPartyName                    [5]     EDIPartyName,
        uniformResourceIdentifier       [6]     IA5String,
        iPAddress                       [7]     OCTET STRING,
        registeredID                    [8]     OBJECT IDENTIFIER }

EDIPartyName ::= SEQUENCE {
        nameAssigner            [0]     DirectoryString OPTIONAL,
        partyName               [1]     DirectoryString }
```

Each GeneralName in the GeneralNames SEQUENCE which encodes a role shall be a directoryName, which is of type Name. Name is an X.501 Distinguished Name. Each Name shall contain exactly one CN (Common Name) component, and zero or one OU (Organizational Unit) components. The OU component, if present, shall specify the authority that defined the semantics of the role. If the OU component is absent, the certificate issuer has defined the role. The CN component shall encode the role ID. Other GeneralName types in the SEQUENCE may be present, but shall not be interpreted as roles. Therefore, if the certificate issuer includes non-role names in the subjectAltName extension, the extension should not be marked critical.

Note that the role, and authority need to be encoded as ASN.1 PrintableString type, the restricted character set [0-9a-z-A-z '()+,-./:=?].

- Key Usage (4.2.1.3)

The key usage extension defines the purpose (e.g., encipherment, signature, certificate signing) of the key contained in the certificate. The usage restriction might be employed when a key that could be used for more than one operation is to be restricted.

This specification does not modify the referenced definition of this extension.

- Basic Constraints (4.2.1.9)

  The basic constraints extension identifies whether the subject of the certificate is a CA and the maximum depth of valid certification paths that include this certificate. Without this extension, a certificate cannot be an issuer of other certificates.

  This specification does not modify the referenced definition of this extension.

- Extended Key Usage (4.2.1.12)

  Extended Key Usage describes allowed purposes for which the certified public key may can be used. When a Device receives a certificate, it determines the purpose based on the context of the interaction in which the certificate is presented, and verifies the certificate can be used for that purpose.

  This specification makes the following modifications to the referenced definition of this extension:

  CAs SHOULD mark this extension as critical.

  CAs MUST NOT issue certificates with the anyExtendedKeyUsage OID (2.5.29.37.0).

  The list of OCF-specific purposes and the assigned OIDs to represent them are:

• Identity certificate	1.3.6.1.4.1.44924.1.6
• Role certificate	1.3.6.1.4.1.44924.1.7

9.3.2.3 Cipher Suite for Authentication, Confidentiality and Integrity

All Devices support the certificate based key management shall support TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 cipher suite as defined in [RFC7251]. To establish a secure channel between two Devices the ECDHE_ECDSA (i.e. the signed version of Diffie-Hellman key agreement) key agreement protocol shall be used. During this protocol the two parties authenticate each other. The confidentiality of data transmission is provided by AES_128_CCM_8. The integrity of data transmission is provided by SHA256. Details are defined in [RFC7251] and referenced therein.

To do lightweight certificate processing, the values of the following fields shall be chosen as follows:

- `signatureAlgorithm` := ANSI X9.62 ECDSA algorithm with SHA256,
- `signature` := ANSI X9.62 ECDSA algorithm with SHA256,
- `subjectPublicKeyInfo` := ANSI X9.62 ECDSA algorithm with SHA256 based on secp256r1 curve.

The certificate `validity` period is a period of time, the CA warrants that it will maintain information about the status of the certificate during the time; this information field is represented as a `SEQUENCE` of two dates:

- the date on which the certificate validity period begins (`notBefore`)

- the date on which the certificate validity period ends (`notAfter`).

Both `notBefore` and `notAfter` should be encoded as `UTCTime`.

The field issuer and subject identify the entity that has signed and issued the certificate and the owner of the certificate. They shall be encoded as UTF8String and inserted in CN attribute.

9.3.2.4 Encoding of Certificate

The ASN.1 distinguished encoding rules (DER) as defined in [ISO/IEC 8825-1] shall be used to encode certificates.

9.3.3 CRL Format

An OCF CRL format is based on [RFC5280], but optional fields are not supported and signature-related fields are optional.

9.3.3.1 CRL Profile and Fields

The OCF CRL shall support the following fields; `signature, issuer, this Update, revocationDate, signaturealgorithm and signatureValue`

- `signature`: the algorithm identifier for the algorithm used by the CA to sign this CRL
- `issuer` : the entity that has signed or issued CRL.
- `this Update` : the issue date of this CRL
- `userCertificate` : certificate serial number
- `revocationDate` : revocation date time
- `signatureAlgorithm`: the cryptographic algorithm used by the CA to sign this CRL
- `signatureValue`: the digital signature computed upon the ASN.1 DER encoded `OCFtbsCertList` (this signature value is encoded as a BIT STRING.)

The signature-related fields such as `signature, signatureAlgorithm, signatureValue` are optional.

```
CertificateList  ::=  SEQUENCE  {
       OCFtbsCertList        TBSCertList,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING
 }
OCFtbsCertList:: = SEQUENCE {
  signature            AlgorithmIdentifier OPTIONAL,
  issuer               Name,
  this Update          Time,
  revokedCertificates RevokedCertificates,
  signatureAlgorithm  AlgorithmIdentifier OPTIONAL,
  signatureValue      BIT STRING OPTIONAL
}
RevokedCertificates     SEQUENCE OF SEQUENCE  {
    userCertificate     CertificateSerialNumber,
    revocationDate      Time
}
```

CRL fields			Description	OCF	X.509
OCFtbsCertList	version		Version v2	Not supported	Optional
	signature		AlgorithmIdentifier	1.2.840.10045.4.3.2(ECDSA algorithm with SHA256,Optional)	Specified in [RFC3279], [RFC4055], and [RFC4491] list OIDs
	issuer		Name	Mandatory	Mandatory
	thisUpdate		Time	Mandatory	Mandatory
	nextUpdate		Time	Not supported	Optional
	revokedCertificates	userCertificate	Certificate Serial Number	Mandatory	Mandatory
		revocationDate	Time	Mandatory	Mandatory
		crlEntryExtentions	Time	Not supported	Optional
	crlExtensions		Extensions	Not supported	Optional
signatureAlgorithm			AlgorithmIdentifier	1.2.840.10045.4.3.2(ECDSA algorithm with SHA256,Optional)	Specified in [RFC3279], [RFC4055], and [RFC4491] list OIDs
signatureValue			BIT STRING	Optional	Mandatory

Table 17 – Comparison between OCF and X.509 CRL fields

9.3.3.2 Encoding of CRL

The ASN.1 distinguished encoding rules (DER method of encoding) defined in [ISO/IEC 8825-1] shall be used to encode CRL.

9.3.4 Resource Model

Device certificates and private keys are kept in cred Resource. CRL is maintained and updated with a separate crl Resource that is defined for maintaining the revocation list.

The `cred` Resource contains the certificate information pertaining to the Device. The `PublicData` Property holds the device certificate and CA certificate chain. `PrivateData` Property holds the Device private key paired to the certificate. (See Section 13.2 for additional detail regarding the /oic/sec/cred Resource).

A certificate revocation list Resource is used to maintain a list of revoked certificates obtained through the CMS. The Device must consider revoked certificates as part of certificate path verification. If the CRL Resource is stale or there are insufficient Platform Resources to maintain a full list, the Device must query the CMS for current revocation status. (See Section 13.3 for additional detail regarding the /oic/sec/crl Resource).

9.3.5 Certificate Provisioning

The CMS (e.g. a hub or a smart phone) issues certificates for new Devices. The CMS shall have its own certificate and key pair. The certificate is either a) self-signed if it acts as Root CA or b) signed by the upper CA in its trust hierarchy if it acts as Sub CA. In either case, the certificate shall have the format described in Section 9.3.2.

The CA in the CMS shall retrieve a Device's public key and proof of possession of the private key, generate a Device's certificate signed by this CA certificate, and then the CMS shall transfer them to the Device including its CA certificate chain. Optionally, the CMS may also transfer one or more role certificates, which shall have the format described in Section 9.3.2. The subjectPublicKey of each role certificate shall match the subjectPublicKey in the Device certificate.

In the below sequence, the Certificate Signing Request (CSR) is defined by PKCS#10 in RFC 2986, and is included here by reference.

The sequence flow of a certificate transfer for a Client-directed model is described in Figure 31.

1. The CMS retrieves a CSR from the Device that requests a certificate. In this CSR, the Device shall place its requested UUID into the subject and its public key in the SubjectPublicKeyInfo. The Device determines the public key to present; this may be an already-provisioned key it has selected for use with authentication, or if none is present, it may generate a new key pair internally and provide the public part. The key pair shall be compatible with the allowed ciphersuites listed in Section 9.3.2.3 and 11.2.3, since the certificate will be restricted for use in OCF authentication.

   If the Device does not have a pre-provisioned key pair and is unable to generate a key pair on its own, then it is not capable of using certificates. The Device shall advertise this fact both by setting the 0x8 bit position in the sct property of /oic/sec/doxm to 0, and return an error that the /oic/sec/csr resource does not exist.

2. The CMS shall transfer the issued certificate and CA chain to the designated Device using the same credid, to maintain the association with the private key. The credential type (oic.sec.cred) used to transfer certificates in Figure 31 is also used to transfer role certificates, by including multiple credentials in the POST from CMS to Device. Identity certificates shall be stored with the credusage property set to `oic.sec.cred.cert' and role certificates shall be stored with the credusage property set to `oic.sec.cred.rolecert'.

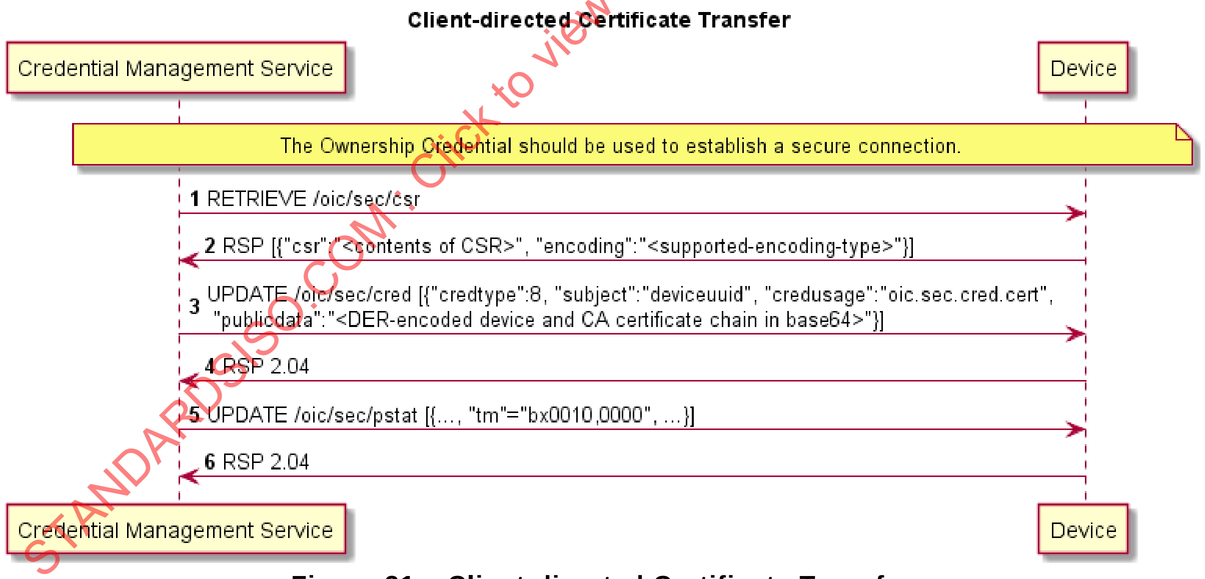

Client-directed Certificate Transfer
Credential Management Service
Device
The Ownership Credential should be used to establish a secure connection.
1 RETRIEVE /oic/sec/csr
2 RSP [{"csr":"<contents of CSR>", "encoding":"<supported-encoding-type>"}]
3 UPDATE /oic/sec/cred [{"credtype":8, "subject":"deviceuuid", "credusage":"oic.sec.cred.cert", "publicdata":"<DER-encoded device and CA certificate chain in base64>"}]
4 RSP 2.04
5 UPDATE /oic/sec/pstat [{..., "tm"="bx0010,0000", ...}]
6 RSP 2.04
Credential Management Service
Device

Figure 31 – Client-directed Certificate Transfer

9.3.6 CRL Provisioning

The only pre-requirement of CRL issuing is that CMS (e.g. a hub or a smart phone) has the function to register revocation certificates, to sign CRL and to transfer it to Devices.

The CMS sends the CRL to the Device.

Any certificate revocation reasons listed below cause CRL update on each Device.

- change of issuer name
- change of association between Devices and CA
- certificate compromise
- suspected compromise of the corresponding private key

CRL may be updated and delivered to all accessible Devices in the OCF network. In some special cases, Devices may request CRL to a given CMS.

There are two options to update and deliver CRL;

- CMS pushes CRL to each Device
- each Device periodically requests to update CRL

The sequence flow of a CRL transfer for a Client-directed model is described in Figure 32.

1. The CMS may retrieve the CRL Resource Property.
2. If the Device requests the CMS to send CRL, it should transfer the latest CRL to the Device.

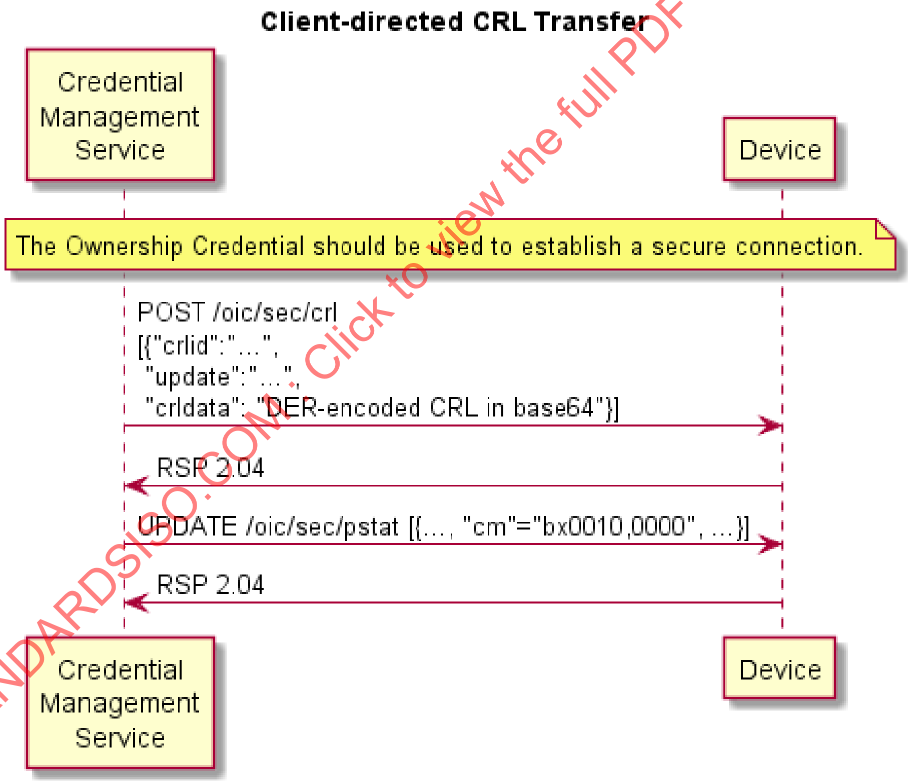

Client-directed CRL Transfer
Credential Management Service
Device
The Ownership Credential should be used to establish a secure connection.
POST /oic/sec/crl
[{"crlid":"...",
"update":"...",
"crldata": "DER-encoded CRL in base64"}]
RSP 2.04
UPDATE /oic/sec/pstat [{..., "cm"="bx0010,0000", ...}]
RSP 2.04
Credential Management Service
Device

Figure 32 – Client-directed CRL Transfer

The sequence flow of a CRL transfer for a Server-directed model is described in Figure 33.

1. The Device retrieves the CRL Resource Property tupdate to the CMS.
2. If the CMS recognizes the updated CRL information after the designated tupdate time, it may transfer its CRL to the Device.

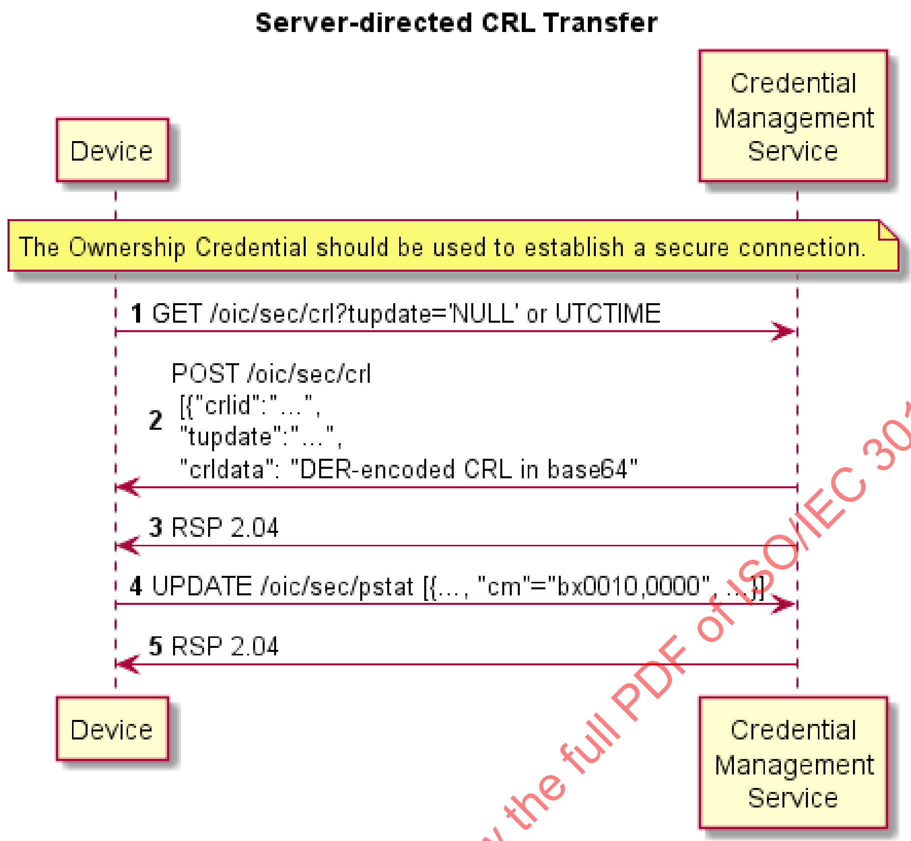

Server-directed CRL Transfer
Device
Credential Management Service
The Ownership Credential should be used to establish a secure connection.
1 GET /oic/sec/crl?tupdate='NULL' or UTCTIME
2 POST /oic/sec/crl [{"crlid":"...", "tupdate":"...", "crldata": "DER-encoded CRL in base64"
3 RSP 2.04
4 UPDATE /oic/sec/pstat [{..., "cm"="bx0010,0000", ...}]
5 RSP 2.04
Device
Credential Management Service

Figure 33 – Server-directed CRL Transfer

10 Device Authentication

When a Client is accessing a restricted Resource on a Server, the Server shall authenticate the Client. Clients shall authenticate Servers while requesting access. Clients may also assert one or more roles that the server can use in access control decisions. Roles may be asserted when the Device authentication is done with certificates.

10.1 Device Authentication with Symmetric Key Credentials

When using symmetric keys to authenticate, the Server Device shall include the ServerKeyExchange message and set psk_identity_hint to the Server's Device ID. The Client shall validate that it has a credential with the Subject ID set to the Server's Device ID, and a credential type of PSK. If it does not, the Client shall respond with an unknown_psk_identity error or other suitable error.

If the Client finds a suitable PSK credential, it shall reply with a ClientKeyExchange message that includes a psk_identity_hint set to the Client's Device ID. The Server shall verify that it has a credential with the matching Subject ID and type. If it does not, the Server shall respond with an unknown_psk_identity or other suitable error code. If it does, then it shall continue with the DTLS protocol, and both Client and Server shall compute the resulting premaster secret.

10.2 Device Authentication with Raw Asymmetric Key Credentials

When using raw asymmetric keys to authenticate, the Client and the Server shall include a suitable public key from a credential that is bound to their Device. Each Device shall verify that the provided public key matches the PublicData field of a credential they have, and use the corresponding Subject ID of the credential to identify the peer Device.

10.3 Device Authentication with Certificates

When using certificates to authenticate, the Client and Server shall each include their certificate chain, as stored in the appropriate credential, as part of the selected authentication cipher suite. Each Device shall validate the certificate chain presented by the peer Device. Each certificate signature shall be verified until a public key is found within the /oic/sec/cred Resource with the `oic.sec.cred.trustca' credusage. Credential Resource found in /oic/sec/cred are used to terminate certificate path validation. Also validity period and revocation status should be checked for all above certificates.

Devices must follow the certificate path validation algorithm in Section 6 of RFC 5280. In particular:

- For all non-end-entity certificates, Devices shall verify that the basic constraints extension is present, and that the cA boolean in the extension is TRUE. If either is false, the certificate chain MUST be rejected. If the pathLenConstraint field is present, Devices will confirm the number of certificates between this certificate and the end-entity certificate is less than or equal to pathLenConstraint. In particular, if pathLenConstraint is zero, only an end-entity certificate can be issued by this certificate. If the pathLenConstraint field is absent, there is no limit to the chain length.

- For all non-end-entity certificates, Devices shall verify that the key usage extension is present, and that the keyCertSign bit is asserted.

- Devices may use the Authority Key Identifier extension to quickly locate the issuing certificate. Devices MUST NOT reject a certificate for lacking this extension, and must instead attempt validation with the public keys of possible issuer certificates whose subject name equals the issuer name of this certificate.

- The end-entity certificate of the chain shall be verified to contain an Extended Key Usage (EKU) suitable to the purpose for which it is being presented. An end-entity certificate which contains no EKU extension is not valid for any purpose and must be rejected. Any certificate which contains the anyExtendedKeyUsage OID (2.5.29.37.0) must be rejected, even if other valid EKUs are also present.

- Devices MUST verify "transitive EKU" for certificate chains. Issuer certificates (any certificate that is not an end-entity) in the chain MUST all be valid for the purpose for which the certificate chain is being presented. An issuer certificate is valid for a purpose if it contains an EKU extension and the EKU OID for that purpose is listed in the extension, OR it does not have an EKU extension. An issuer certificate SHOULD contain an EKU extension and a complete list of EKUs for the purposes for which it is authorized to issue certificates. An issuer certificate without an EKU extension is valid for all purposes; this differs from end-entity certificates without an EKU extension.

The list of purposes and their associated OIDs are defined in Section 9.3.2.2.

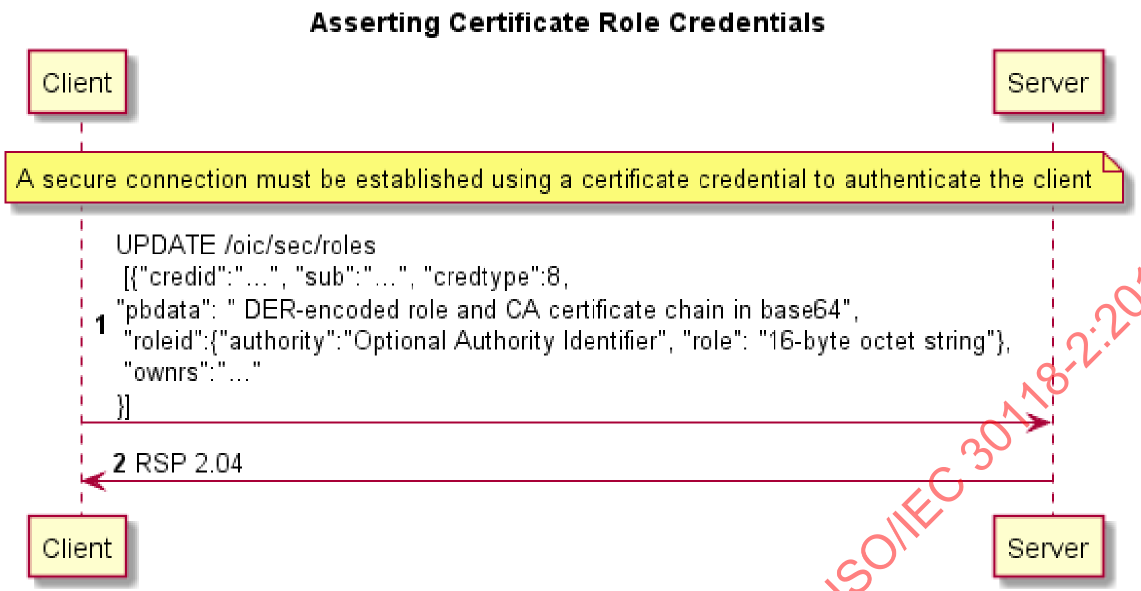
If the Device does not recognize an extension, it must examine the `critical` field. If the field is TRUE, the Device MUST reject the certificate. If the field is FALSE, the Device MUST treat the certificate as if the extension were absent and proceed accordingly. This applies to all certificates in a chain.

Note: Certificate revocation mechanisms are currently out of scope of this version of the specification.

10.3.1 Role Assertion with Certificates

This section describes role assertion by a client to a server using a certificate role credential. If a server does not support the certificate credential type, clients should not attempt to assert roles with certificates.

Following authentication with a certificate, a client may assert one or more roles by updating the server's roles resource with the role certificates it wants to use. The role credentials must be certificate credentials and shall include a certificate chain. The server shall validate each certificate chain as specified in Section 10.3. Additionally, the public key in the end-entity certificate used for Device authentication must be identical to the public key in all role (end-entity) certificates. Also, the subject distinguished name in the end-entity authentication and role certificates must match. The roles asserted are encoded in the subjectAltName extension in the certificate. Note that the subjectAltName field can have multiple values, allowing a single certificate to encode multiple roles that apply to the client. The server shall also check that the EKU extension of the role certificate(s) contains the value 1.3.6.1.4.1.44924.1.7 (see Section 9.3.2.1) indicating the certificate may be used to assert roles. Figure 34 describes how a client Device asserts roles to a server.


Asserting Certificate Role Credentials
Client
Server
A secure connection must be established using a certificate credential to authenticate the client
1
2 RSP 2.04
Client
Server

```
UPDATE /oic/sec/roles
 [{"credid":"...", "sub":"...", "credtype":8,
"pbdata": " DER-encoded role and CA certificate chain in base64",
 "roleid":{"authority":"Optional Authority Identifier", "role": "16-byte octet string"},
 "ownrs":"..."
}]
```

Figure 34 – Asserting a role with a certificate role credential.

Figure 34 Notes

1. The response shall contain "204 No Content" to indicate success or 4xx to indicate an error. If the server does not support certificate credentials, it should return "501 Not Implemented"

2. Roles asserted by the client may be kept for a duration chosen by the server. The duration shall not exceed the validity period of the role certificate. When fresh CRL information is obtained, the certificates in /oic/sec/roles should be checked, and the role removed if the certificate is revoked or expired.

   Servers should choose a nonzero duration to avoid the cost of frequent re-assertion of a role by a client. It is recommended that servers use the validity period of the certificate as a duration, effectively allowing the CMS to decide the duration.

3. The format of the data sent in the create call shall be a list of credentials (oic.sec.cred, see Table 23). They shall have credtype 8 (indicating certificates) and PrivateData field shall not be present. For fields that are duplicated in the oic.sec.cred object and the certificate, the value in the certificate shall be used for validation. For example, if the Period field is set in the credential, the server amust treat the validity period in the certificate as authoritative. Similar for the roleid data (authority, role).

4. Certificates shall be encoded as in Figure 31 (DER-encoded certificate chain in base64)

5. Clients may GET the /oic/sec/roles resource to determine the roles that have been previously asserted. An array of credential objects must be returned, or "204 No Content" to indicate that no previously asserted roles are currently valid.

11 Message Integrity and Confidentiality

Secured communications between Clients and Servers are protected against eavesdropping, tampering, or message replay, using security mechanisms that provide message confidentiality and integrity.

11.1 Session Protection with DTLS

Devices shall support DTLS for secured communications as defined in [RFC 6347]. Devices using TCP shall support TLS v1.2 for secured communications as defined in [RFC 5246]. See Section 11.2 for a list of required and optional cipher suites for message communication.

OCF Devices MUST support (D)TLS version 1.2 or greater and MUST NOT support versions 1.1 or lower.

Note: Multicast session semantics are not yet defined in this version of the security specification.

11.1.1 Unicast Session Semantics

For unicast messages between a Client and a Server, both Devices shall authenticate each other. See Section 10 for details on Device Authentication.

Secured unicast messages between a Client and a Server shall employ a cipher suite from Section 11.2. The sending Device shall encrypt and authenticate messages as defined by the selected cipher suite and the receiving Device shall verify and decrypt the messages before processing them.

11.2 Cipher Suites

The cipher suites allowed for use can vary depending on the context. This section lists the cipher suites allowed during ownership transfer and normal operation. The following RFCs provide additional information about the cipher suites used in OCF.

[RFC 4279]: Specifies use of pre-shared keys (PSK) in (D)TLS
[RFC 4492]: Specifies use of elliptic curve cryptography in (D)TLS
[RFC 5489]: Specifies use of cipher suites that use elliptic curve Diffie-Hellman (ECDHE) and PSKs
[RFC 6655, 7251]: Specifies AES-CCM mode cipher suites, with ECDHE

11.2.1 Cipher Suites for Device Ownership Transfer

11.2.1.1 Just Works Method Cipher Suites

The Just Works owner transfer method may use the following (D)TLS cipher suites.
TLS_ECDH_ANON_WITH_AES_128_CBC_SHA256,
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA256

All Devices supporting Just Works OTM shall implement:
TLS_ECDH_ANON_WITH_AES_128_CBC_SHA256 (with the value 0xFF00)

All Devices supporting Just Works OTM should implement:
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA256 (with the value 0xFF01)

11.2.1.2 Random PIN Method Cipher Suites

The Random PIN Based owner transfer method may use the following (D)TLS cipher suites.
TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,

All Devices supporting Random Pin Based OTM shall implement:
TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256

11.2.1.3 Certificate Method Cipher Suites

The Manufacturer Certificate Based owner transfer method may use the following (D)TLS cipher suites.

    TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8,
    TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8,
    TLS_ECDHE_ECDSA_WITH_AES_128_CCM,
    TLS_ECDHE_ECDSA_WITH_AES_256_CCM

Using the following curve:

    secp256r1 (See [RFC4492])

All Devices supporting Manufacturer Certificate Based OTM shall implement:

    TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

Devices supporting Manufacturer Certificate Based OTM should implement:

    TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8,
    TLS_ECDHE_ECDSA_WITH_AES_128_CCM,
    TLS_ECDHE_ECDSA_WITH_AES_256_CCM

11.2.2 Cipher Suites for Symmetric Keys

The following cipher suites are defined for (D)TLS communication using PSKs:

    TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
    TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
    TLS_PSK_WITH_AES_128_CCM_8, (* 8 OCTET Authentication tag *)
    TLS_PSK_WITH_AES_256_CCM_8,
    TLS_PSK_WITH_AES_128_CCM, (* 16 OCTET Authentication tag *)
    TLS_PSK_WITH_AES_256_CCM,

Note: All CCM based cipher suites also use HMAC-SHA-256 for authentication.

All Devices shall implement the following:

    TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,

Devices should implement the following:

    TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
    TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
    TLS_PSK_WITH_AES_128_CCM_8,
    TLS_PSK_WITH_AES_256_CCM_8,
    TLS_PSK_WITH_AES_128_CCM,
    TLS_PSK_WITH_AES_256_CCM

11.2.3 Cipher Suites for Asymmetric Credentials

The following cipher suites are defined for (D)TLS communication with asymmetric keys or certificates:

    TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8,
    TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8,
    TLS_ECDHE_ECDSA_WITH_AES_128_CCM,
    TLS_ECDHE_ECDSA_WITH_AES_256_CCM

Using the following curve:

    secp256r1 (See [RFC4492])

All Devices supporting Asymmetric Credentials shall implement:

    TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

All Devices supporting Asymmetric Credentials should implement:

    TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8,
    TLS_ECDHE_ECDSA_WITH_AES_128_CCM,
    TLS_ECDHE_ECDSA_WITH_AES_256_CCM

12 Access Control

12.1 ACL Generation and Management

This section will be expanded in a future version of the specification.

12.2 ACL Evaluation and Enforcement

The Server enforces access control over application Resources before exposing them to the requestor. The Security Resource Manager (SRM) in the Server authenticates the requestor when access is received via the secure port. Authenticated requestors, known as the “subject” can be used to match ACL entries that specify the requestor’s identity, role or may match authenticated requestors using a subject wildcard.

If the request arrives over the unsecured port, the only ACL policies allowed are those that use a subject wildcard match of anonymous requestors.

Access is denied if a requested resource is not matched by an ACL entry. (Note: There are documented exceptions pertaining to Device onbording where access to security virtual resources may be permitted prior to provisioning of ACL resources.

The second generation ACL (i.e. /oic/sec/acl2) contains an array of Access Control Entries (ACE2) that employ a resource matching algorithm that uses an array of resource references to match Resources to which the ACE2 access policy applies. Matching consists of comparing the values of the ACE2 "resources" property (see Section 13) to the requested Resource. Resources are matched in four ways; host reference (href), resource type (rt), resource interface (if) or resource wildcard.

12.2.1 Host Reference Matching

When present in an ACE2 matching element, the Host Reference (href) Property shall be used for resource matching.

- The href Property shall be used to find an exact match of the Resource name.

12.2.2 Resource Type Matching

When present in an ACE2 matching element, The Resource Type (rt) Property shall be used for resource matching.

- The rt Property shall be used to find an exact match of the Resource Type name.
- An array of strings is used to match Resources that implement multiple Resource Type names (e.g. collection resources).

12.2.3 Interface Matching

When present in the ACE2 matching element, the Interface (if) property shall be used for resource matching.

- The ‘if’ property shall be used to find an exact match of the Resource Interface string.
- An array of strings is used when the Resource implements multiple Interfaces.

12.2.4 Multiple Criteria Matching

If multiple matching criteria are supplied in the same ACE2 Resources property (e.g. ‘href’ and ‘rt’ and ‘if’) then a logical AND of the criteria shall be applied. For example, if both ‘href’="/a/light and ‘if’="oic.if.s" are in the Resources property, then a match exists only when both the ‘href’ and the ‘if’ criterion are true for the candidate resources.

If the ACE2 "resources" property is an array of entries, then a logical OR is applied for each array element. For example, if a first array element of the Resources property contains ‘href’="/a/light" and the second array element of the Resources property contains ‘if’="oic.if.s", then Resources that match either the ‘href’ criteria or the ‘if’ criteria are included in the set of matched Resources.

12.2.5 Resource Wildcard Matching

A wildcard expression may be used to match multiple Resources using a wildcard Property contained in the oic.sec.ace2.resource-ref structure. The following wildcard matching strings are defined:

String	Description
"+"	Shall match all discoverable resources.
"-"	Shall match all non-discoverable resources.
"*"	Shall match all resources.

Table 18 – ACE2 Wildcard Matching Strings Description

Note: Discoverable resources appear in the /oic/wk/res Resource, while non-discoverable resources may appear in other collection resources but do not appear in the /res collection.

Example JSON for Resource matching

```
{
  [
//Matches Resources named "/x/door1" or "/x/door2"
     {
         "href":"/x/door1"
     },
     {
         "href":"/x/door2"
     },
//Matches Resources with Resource Type "oic.sec.crl" and "oic.sec.cred"
     {
         "rt":[" oic.sec.crl ", "oic.sec.cred "]
     },
// Matches Resources that implement both "oic.if.baseline" and
"oic.if.rw" Interfaces.

         "if":["oic.if.baseline", "oic.if.rw"]
     },
//Matches Resources named "/x/light1" or "/x/light2" and have Resource
Types "x.light.led", "x.light.flourescent" and "x.light.color".
     {
         "href":"/x/light1",
         "rt":["x.light.led","x.light.flourescent", "x.light.color"]
     },
     {
         "href":"/x/light2",
         "rt":["x.light.led","x.light.flourescent", "x.light.color"]
     },
//Matches all Resources.
     {
      "wc":"*"
     }
  ]
}
```

12.2.6 Subject Matching using Wildcards

When the ACE subject is specified as the wildcard string "*" any requestor is matched. The OCF server may authenticate the OCF client, but is not required to.

Examples: JSON for subject wildcard matching

```
//matches all subjects that have authenticated and confidentiality
protections in place.
"subject" : {
   "conntype" : "auth-crypt"
}
//matches all subjects that have NOT authenticated and have NO
confidentiality protections in place.
"subject" : {
    "conntype" : "anon-clear"
}
```

12.2.7 Subject Matching using Roles

When the ACE subject is specified as a role, a requestor shall be matched if either:

1. The requestor authenticated with a symmetric key credential, and the role is present in the roleid property of the credential's entry in the credential resource, or
2. The requestor authenticated with a certificate, and a valid role certificate is present in the roles resource with the requestor's certificate's public key at the time of evaluation. Validating role certificates is defined in section 10.3.1.

12.2.8 ACL Evaluation

The OCF Server shall apply an ACE2 matching algorithm that matches in the following sequence:

1. If the /oic/sec/sacl Resource exists and if the signature verification is successful, these ACE2 entries contribute to the set of local ACE2 entries in step 3. The Server shall verify the signature, at least once, following update of the /oic/sec/sacl Resource.
2. The local /oic/sec/acl2 Resource contributes its ACE2 entries for matching.
3. Access shall be granted when all these criteria are met:
    a. The requestor is matched by the ACE2 "subject" Property.
    b. The requested Resource is matched by the ACE2 "resources" Property and the requested Resource shall exist on the local Server.
    c. The "period" Property constraint shall be satisfied.
    d. The "permission" Property constraint shall be applied.

Note: If multiple ACE2 entries match the Resource request, the union of permissions, for all matching ACEs, defines the *effective* permission granted. E.g. If Perm1=CR---; Perm2=--UDN; Then UNION (Perm1, Perm2)=CRUDN.

The Server shall enforce access based on the effective permissions granted.

13 Security Resources

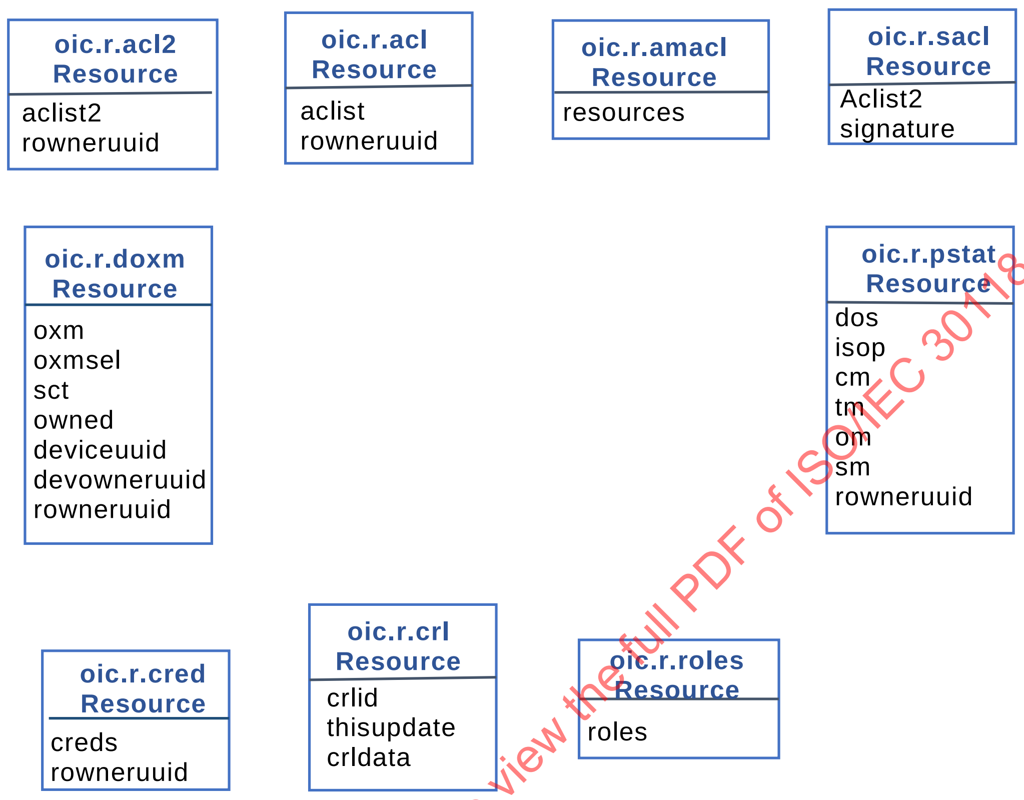


Figure 35 – OCF Security Resources

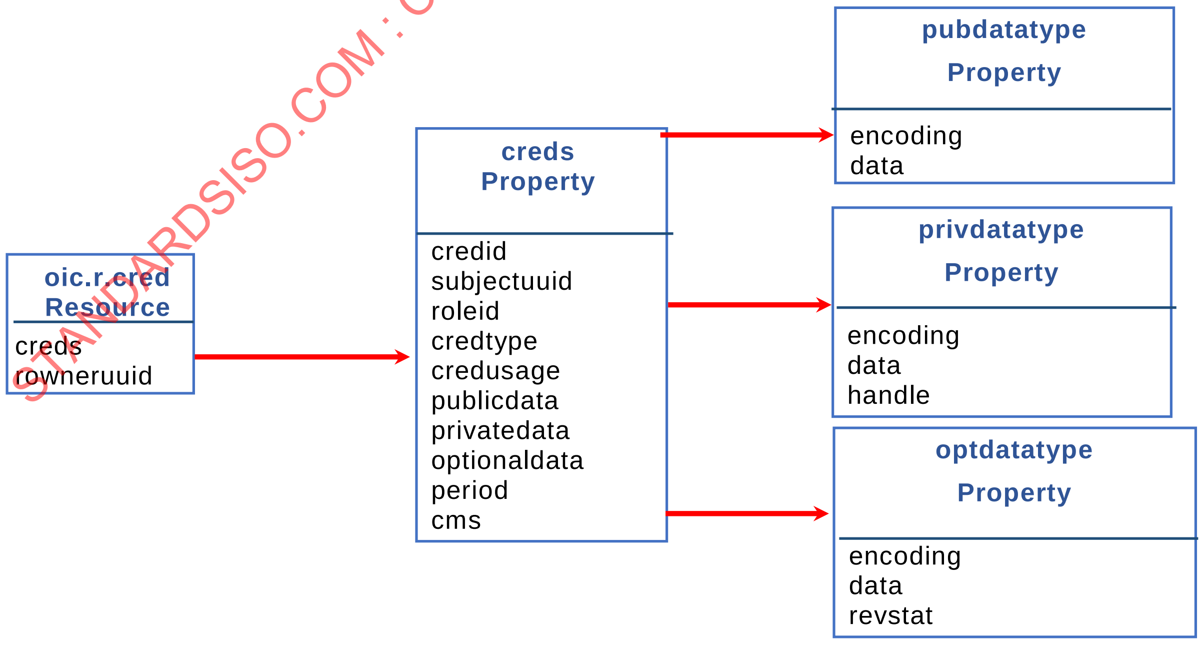


Figure 36 – oic.r.cred Resource and Properties

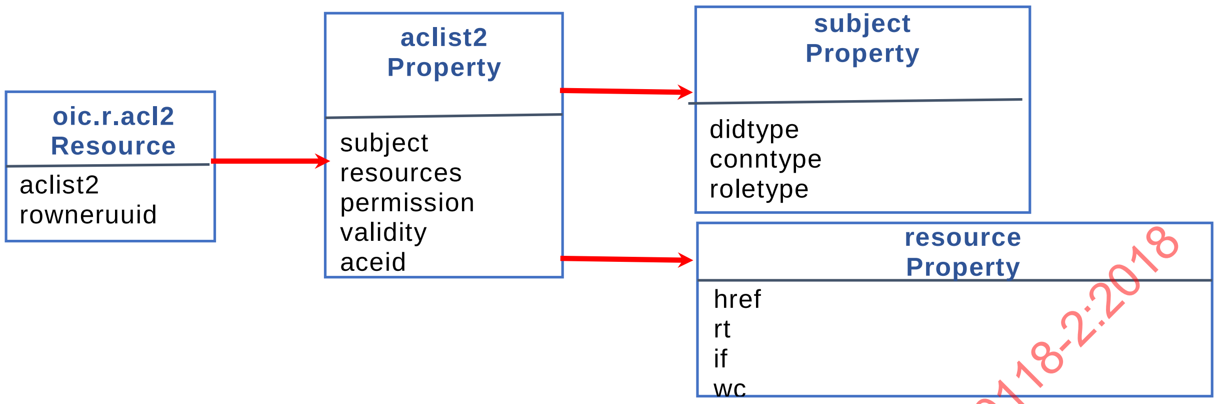


Figure 37 – oic.r.acl2 Resource and Properties

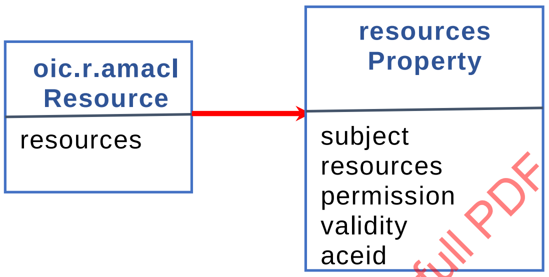


Figure 38 – oic.r.amacl Resource and Properties

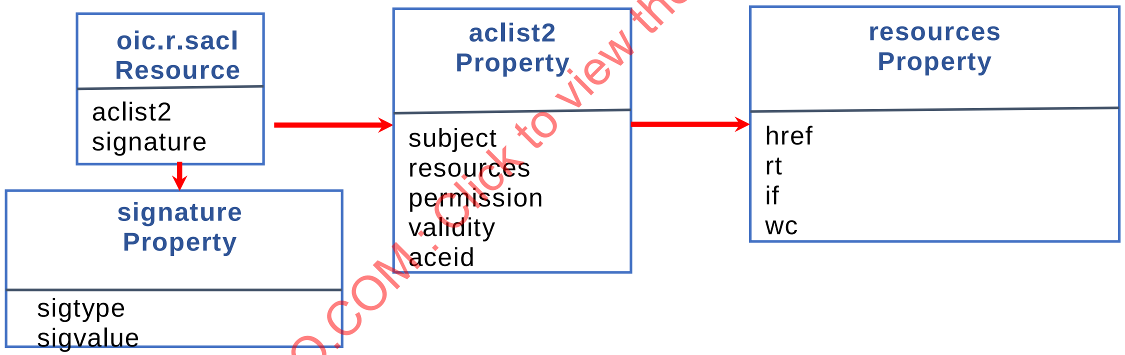


Figure 39 – oic.secr.sacl Resource and Properties

13.1 Device Owner Transfer Resource

The /oic/sec/doxm Resource contains the set of supported Device owner transfer methods.

Resource discovery processing respects the CRUDN constraints supplied as part of the security Resource definitions contained in this specification.

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/doxm	Device Owner Transfer Methods	urn:oic.r.doxm	baseline	Resource for supporting Device owner transfer	Configuration

Table 19 – Definition of the oic.r.doxm Resource

Property Title	Property Name	Value Type	Value Rule	Mandatory	Device State	Access Mode	Description
Owner Transfer Method	oxms	oic.sec.doxmtype	array	Yes		R	Value identifying the owner-transfer-method and the organization that defined the method.
Oxm Selection	oxmsel	oic.sec.doxmtype	UINT16	Yes	RESET	R	Server shall set to (4) "oic.sec.oxm.self"
					RFOTM	RW	The as yet unauthenticated DOXS shall set to its selected OTM and both parties execute the OTM. After secure owner transfer session is established, DOXS shall update the oxmsel again making it permanent. If the OTM fails the Server shall transition Device state to RESET.
					RFPRO	R	n/a
					RFNOP	R	n/a
					SRESET	R	n/a
Supported Credential Types	sct	oic.sec.credtype	bitmask	Yes		R	Identifies the types of credentials the Device supports. The SRM sets this value at framework initialization after determining security capabilities.
Owned	owned	Boolean	T|F	Yes	RESET	R	Server shall set to FALSE.
					RFOTM	RW	DOXS shall set to TRUE after secure owner transfer session is established.
					RFPRO	R	n/a
					RFNOP	R	n/a
					SRESET	R	n/a
Device UUID	deviceuuid	String	oic.sec.didtype	Yes	RESET	R	Server shall construct a temporary random UUID that differs for each transition to RESET.
					RFOTM	RW	DOXS shall update to a value it has selected after secure owner transfer session is established. If update fails with error PROPERTY_NOT_FOUND the DOXS shall either accept the Server provided value or update /doxm.owned=FALSE and terminate the session.
					RFPRO	R	n/a
					RFNOP	R	n/a
					SRESET	R	n/a
Device Owner Id	devowneruuid	String	uuid	Yes	RESET	R	Server shall set to the nil uuid value (e.g. "00000000-0000-0000-0000-000000000000")

					RFOTM	RW	DOXS shall set value after secure owner transfer session is established.
					RFPRO	R	n/a
					RFNOP	R	n/a
					SRESET	R	n/a
Resource Owner Id	rowneruuid	String	uuid	Yes	RESET	R	Server shall set to the nil uuid value (e.g. "00000000-0000-0000-0000-000000000000")
					RFOTM	RW	The DOXS should configure the rowneruuid property when a successful owner transfer session is established.
					RFPRO	R	n/a
					RFNOP	R	n/a
					SRESET	RW	The DOXS (referenced via /doxm.devowneruuid property) should verify and if needed, update the resource owner property when a mutually authenticated secure session is established. If the rowneruuid does not refer to a valid DOXS the Server shall transition to RESET Device state.

Table 20 – Properties of the oic.r.doxm Resource

Property Title	Propert y Name	Value Type	Value Rule	Man dato ry	Device State	Access Mode	Description
Device ID	uuid	String	uuid	Yes	RW	-	A uuid value

Table 21 - Properties of the oic.sec.didtype Property

The owner transfer method (oxms) Property contains a list of owner transfer methods where the entries appear in the order of preference. The Device manufacturer configures this Property with the most desirable methods appearing before the lower priority methods. The network management tool queries this list at the time of onboarding when the network management tool selects the most appropriate method.

Subsequent to an owner transfer method being chosen the agreed upon method shall be entered into the /doxm Resource using the oxmsel Property.

Owner transfer methods consist of two parts, a URN identifying the vendor or organization and the specific method.

<OxmType> ::= "urn:" <NID> ":" <NSS>

<NID> :: = <Vendor-Organization>

<NSS> ::= <Method> | {<NameSpaceQualifier> "."} <Method>

<NameSpaceQualifier> ::= String

<Method> ::= String

<Vendor-Organization> ::= String

When an owner transfer method successfully completes, the *owned* Property is set to '1' (TRUE). Consequently, subsequent attempts to take ownership of the Device will fail.

The SRM generates a Device identifier (deviceuuid) that is stored in the /oic/sec/doxm Resource in response to successful ownership transfer.

Owner transfer methods should communicate the deviceuuid to the service that is taking ownership. The service should associate the deviceuuid with the OC in a secured database.

The Device vendor shall determine that the Device identifier (deviceuuid) is persistent (not updatable) or that it is non-persistent (updatable by the owner transfer service – a.k.a DOXS).

If deviceuuid is persistent, the request to update shall fail with the error PROPERTY_NOT_FOUND.

If it is non-persistent, the request to update shall succeed and the value supplied by DOXS shall be remembered until the Device is RESET. If the update fails for any other reason and Device state has not transitioned to RESET, the value of deviceuuid shall be the nil UUID (e.g. "00000000-0000-0000-0000-000000000000").

Regardless of whether the Device has a persistent or non-persistent deviceuuid, a temporal random non-repeating UUID is found each time the Device enters RESET. The temporal deviceuuid is used while the Device state is in the RESET state and while in the RFOTM Device state until the DOXS establishes a secure OTM connection.

13.1.1 OCF defined owner transfer methods

Value Type Name	Value Type URN (optional)	Enumeration Value (mandatory)	Description
OCFJustWorks	oic.sec.doxm.jw	0	The just-works method relies on anonymous Diffie-Hellman key agreement protocol to allow an OBT to assert ownership of the new Device. The first OBT to make the assertion is accepted as the Device owner. The just-works method results in a shared secret that is used to authenticate the Device to the OBT and likewise authenticates the OBT to the Device. The Device allows the OBT to take ownership of the Device, after which a second attempt to take ownership by a different OBT will fail. Note: The just-works method is subject to a man-in-the-middle attacker. Precautions should be taken to provide physical security when this method is used.
OCFSharedPin	oic.sec.doxm.rdp	1	The new Device randomly generates a PIN that is communicated via an out-of-band channel to a Device OBT. An in-band Diffie-Hellman key agreement protocol establishes that both endpoints possess the PIN. Possession of the PIN by the OBT signals the new Device that device ownership can be asserted.
OCFMfgCert	oic.sec. doxm.mfgcert	2	The new Device is presumed to have been manufactured with an embedded asymmetric private key that is used to sign a Diffie-Hellman exchange at Device onboarding. The manufacturer certificate should contain Platform hardening information and other security assurances assertions.
OCF Reserved	<Reserved>	3	Reserved
OCFSelf	oic.sec.oxm.self	4	The manufacturer shall set the /doxm.oxmsel value to (4). The Server shall reset this value to (4) upon entering RESET Device state.
OCF Reserved	<Reserved>	5~0xFEFF	Reserved for OCF use
Vendor-defined Value Type Name	<Reserved>	0xFF00~0xFFFF	Reserved for vendor-specific OTM use

Table 22 – Properties of the oic.sec.doxmtype Property

13.2 Credential Resource

The /oic/sec/cred Resource maintains credentials used to authenticate the Server to Clients and support services as well as credentials used to verify Clients and support services.

Multiple credential types are anticipated by the OCF framework, including pair-wise pre-shared keys, asymmetric keys, certificates and others. The credential Resource uses a Subject UUID to distinguish the Clients and support services it recognizes by verifying an authentication challenge.

In order to provide an interface which allows management of the "creds" Array Property, the RETRIEVE, UPDATE and DELETE operations on the oic.r.cred Resource shall behave as follows:

1. A RETRIEVE shall return the full Resource representation, except that any write-only Properties shall be omitted (e.g. private key data).

2. An UPDATE shall replace or add to the Properties included in the representation sent with the UPDATE request, as follows:

   a. If an UPDATE representation includes the "creds" array Property, then:

      i. Supplied creds with a "credid" that matches an existing "credid" shall replace completely the corresponding cred in the existing "creds" array.

      ii. Supplied creds without a "credid" shall be appended to the existing "creds" array, and a unique (to the cred Resource) "credid" shall be created and assigned to the new cred by the Server. The "credid" of a deleted cred should not be reused, to improve the determinism of the interface and reduce opportunity for race conditions.

      iii. Supplied creds with a "credid" that does not match an existing "credid" shall be appended to the existing "creds" array, using the supplied "credid".

3. A DELETE without query parameters shall remove the entire "creds" array, but shall not remove the oic.r.cred Resource.

4. A DELETE with one or more "credid" query parameters shall remove the cred(s) with the corresponding credid(s) from the "creds" array.

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/cred	Credentials	urn:oic.r.cred	baseline	Resource containing credentials for Device authentication, verification and data protection	Security

Table 23 – Definition of the oic.r.cred Resource

Property Title	Property Name	Value Type	Value Rule	Mandatory	Device State	Access Mode	Description
Credentials	creds	oic.sec.cred	array	Yes	RESET	R	Server shall set to manufacturer defaults.
					RFOTM	RW	Set by DOXS after successful OTM
					RFPRO	R	Set by the CMS (referenced via the /cred.rowneruuid property) after successful authentication. Access to vertical resources is prohibited.
					RFNOP	R	Access to vertical resources is permitted after a matching ACE is found.
					SRESET	RW	The DOXS (referenced via /doxm.devowneruuid property) should evaluate the integrity of and may update creds entries when a secure session is established and the Server and DOXS are authenticated.
Resource Owner ID	rowneruuid	String	uuid	Yes	RESET	R	Server shall set to the nil uuid value (e.g. "00000000-0000-0000-0000-000000000000")
					RFOTM	RW	The DOXS should configure the /cred.owneruuid property when a successful owner transfer session is established.
					RFPRO	R	n/a
					RFNOP	R	n/a
					SRESET	RW	The DOXS (referenced via /doxm.devowneruuid property) should verify and if needed, update the resource owner property when a mutually authenticated secure session is established. If the rowneruuid does not refer to a valid DOXS the Server shall transition to RESET Device state.

Table 24 – Properties of the oic.r.cred Resource

All secure Device accesses shall have a /oic/sec/cred Resource that protects the end-to-end interaction.

The /oic/sec/cred Resource can be created and modified by the services named in the ‘rowneruuid’’ Property.

ACLs naming /oic/sec/cred Resource should further restrict access beyond CRUDN access modes.

Property Title	Property Name	Value Type	Value Rule	Mandatory	Access Mode	Device State	Description
Credential ID	credid	UINT16	0 – 64K-1	Yes	RW		Short credential ID for local references from other Resource
Subject UUID	subjectuuid	String	uuid	Yes	RW		A uuid that identifies the subject to which this credential applies
Role ID	roleid	oic.sec.roletype	-	No	RW		Identifies the role(s) the subject is authorized to assert.
Credential Type	credtype	oic.sec.credtype	bitmask	Yes	RW		Represents this credential's type. 0 – Used for testing 1 – Symmetric pair-wise key 2 – Symmetric group key 4 – Asymmetric signing key 8 – Asymmetric signing key with certificate 16 – PIN or password 32 – Asymmetric encryption key
Credential Usage	credusage	String	-	No	RW		Used to resolve undecidability of the credential. Provides indication for how/where the cred is used oic.sec.cred.trustca: certificate trust anchor oic.sec.cred.cert: identity certificate oic.sec.cred.rolecert: role certificate oic.sec.cred.mfgtrustca: manufacturer certificate trust anchor oic.sec.cred.mfgcert: manufacturer certificate
Public Data	publicdata	oic.sec.pubdatatype	-	No	RW		Public credential information 1:2: ticket, public SKDC values 4, 32: Public key value 8: certificate
Private Data	privatedata	oic.sec.privdatatype	-	No			1:2: symmetric key 4: 8, 32, 64: Private asymmetric key 16: password hash, password value, security questions
					-	RESET	Server shall set to manufacturer default
					W	RFOTM	Set by DOXS after successful OTM
					W	RFPRO	Set by authenticated DOXS or CMS
					-	RFNOP	Not writable during normal operation.
					W	SRESET	DOXS may modify to enable transition to RFPRO.

Optional Data	optionaldata	oic.sec.optdatatype	-	No	RW		Credential revocation status information 1, 2, 4, 32: revocation status information 8: Revocation + CA certificate.
Period	period	String	-	No	RW		Period as defined by RFC5545. The credential should not be used if the current time is outside the Period window.
Credential Refresh Method	crms	oic.sec.crmtype	array	No	RW		Credentials with a Period Property are refreshed using the credential refresh method (crm) according to the type definitions for oic.sec.crm.

Table 25 – Properties of the oic.sec.cred Property

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
Encoding format	encoding	String	-	RW	No	A string specifying the encoding format of the data contained in the pubdata "oic.sec.encoding.jwt" - RFC7517 JSON web token (JWT) encoding "oic.sec.encoding.cwt" - RFC CBOR web token (CWT) encoding "oic.sec.encoding.base64" – Base64 encoding "oic.sec.encoding.uri" – URI reference "oic.sec.encoding.pem" – Encoding for PEM-encoded certificate or chain "oic.sec.encoding.der" – Encoding for DER-encoded certificate or chain "oic.sec.encoding.raw" – Raw hex encoded data
Data	data	String	-	RW	No	The encoded value

Table 26 – Properties of the oic.sec.pubdatatype Property

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
Encoding format	encoding	String	-	RW	Yes	A string specifying the encoding format of the data contained in the privdata "oic.sec.encoding.jwt" - RFC7517 JSON web token (JWT) encoding "oic.sec.encoding.cwt" - RFC CBOR web token (CWT) encoding "oic.sec.encoding.base64" – Base64 encoding "oic.sec.encoding.uri" – URI reference "oic.sec.encoding.handle" – Data is contained in a storage sub-system referenced using a handle "oic.sec.encoding.raw" – Raw hex encoded data
Data	data	String	-	RW	No	The encoded value This value shall never be readable.
Handle	handle	UINT16	-	RW	No	Handle to a key storage resource

Table 27 – Properties of the oic.sec.privdatatype Property

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
Revocation status	revstat	Boolean	T | F	RW	Yes	Revocation status flag True – revoked False – not revoked
Encoding format	encoding	String	-	RW	No	A string specifying the encoding format of the data contained in the optdata "oic.sec.encoding.jwt" - RFC7517 JSON web token (JWT) encoding "oic.sec.encoding.cwt" - RFC CBOR web token (CWT) encoding "oic.sec.encoding.base64" – Base64 encoding "oic.sec.encoding.pem" – Encoding for PEM-encoded certificate or chain "oic.sec.encoding.der" – Encoding for DER-encoded certificate or chain "oic.sec.encoding.raw" – Raw hex encoded data
Data	data	String	-	RW	No	The encoded structure

Table 28 – Properties of the oic.sec.optdatatype Property

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
Authority	authority	String		R	No	A name for the authority that defined the role. If not present, the credential issuer defined the role. If present, must be expressible as an ASN.1 PrintableString.
Role	role	String	-	R	Yes	An identifier for the role. Must be expressible as an ASN.1 PrintableString.

Table 29 – Definition of the oic.sec.roletype Property.

13.2.1 Properties of the Credential Resource

13.2.1.1 Credential ID

Credential ID (credid) is a local reference to a /oic/sec/cred instance. The SRM generates it. credid shall be used to disambiguate Resource instances that have the same Subject UUID.

13.2.1.2 Subject UUID

Subject UUID identifies the Device or service to which a credential Resource shall be used to establish a secure session, verify an authentication challenge-response or to authenticate an authentication challenge.

A Subject UUID that matches the Server's own Device ID identifies credentials that authenticate this Device.

Subject UUID shall be used to identify a group to which a group key is used to protect shared data.

13.2.1.3 Role ID

Role ID identifies the set of roles that have been granted to the Subject UUID. The asserted role or set of roles shall be a subset of the role values contained in the roleid Property.

If a credential contains a set of roles, ACL matching succeeds if the asserted role is a member of the role set in the credential.

13.2.1.4 Credential Type

The Credential Type is used to interpret several of the other Property values whose contents can differ depending on the type of credential. These properties include publicdata, privatedata and optionaldata. The CredType value of ‘0’ ("no security mode") is reserved for testing and debugging circumstances. Production deployments should not allow provisioning of credentials of type ‘0’. The SRM should introduce checking code that prevents its use in production deployments.

13.2.1.5 Public Data

Public Data contains information that provides additional context surrounding the issuance of the credential. For example, it might contain information included in a certificate or response data from a Key Management Service. It might contain wrapped data such as a SKDC issued ticket that has yet to be delivered.

13.2.1.6 Private Data

Private Data contains the secret information that is used to authenticate the Device, protect or unprotect data or verify an authentication challenge-response.

Private Data shall not be disclosed outside of the SRM’s trusted computing base. A secure element (SE) or trusted execution environment (TEE) should be used to implement the SRM’s trusted computing base. In this situation, the Private Data contents should be a handle or reference to secure storage resources.

13.2.1.7 Optional Data

Optional Data contains information that is optionally supplied, but facilitates key management, scalability or performance optimization. For example, if the Credential Type identifies certificates, it contains a certificate revocation status value and the Certificate Authority (CA) certificate that will be used for mutual authentication.

13.2.1.8 Period

The Period Property identifies the validity period for the credential. If no validity period is specified the credential lifetime is undetermined. Constrained Devices that do not implement a date-time capability shall obtain current date-time information from it’s CMS.

13.2.1.9 Credential Refresh Method Type Definition

The oic.sec.crm defines the credential refresh methods that the CMS shall implement.

Value Type Name	Value Type URN	Applicable Credential Type	Description
Provisioning Service	oic.sec.crm.pro	All	A CMS initiates re-issuance of credentials nearing expiration. The Server should delete expired credentials to manage storage resources. The Resource Owner Property references the provisioning service. The Server uses its /oic/sec/cred.rowneruuid Resource to identify additional key management service that supports this credential refresh method.
Pre-shared Key	oic.sec.crm.psk	[1]	The Server performs ad-hoc key refresh by initiating a DTLS connection with the Device prior to credential expiration using a Diffie-Hellman based ciphersuite and the current PSK. The new DTLS MasterSecret value becomes the new PSK. The Server selects the new validity period. The new validity period value is sent to the Device who updates the validity period for the current credential. The Device acknowledges this update by returning a successful response or denies the update by returning a failure response. The Server uses its /oic/sec/cred.rowneruuid Resource to identify a key management service that supports this credential refresh method.
Random PIN	oic.sec.crm.rdp	[16]	The Server performs ad-hoc key refresh following the oic.sec.crm.psk approach, but in addition generates a random PIN value that is communicated out-of-band to the remote Device. The current PSK + PIN are hashed to form a new PSK' that is used with the DTLS ciphersuite. I.e. PSK' = SHA256(PSK, PIN). The Server uses its /oic/sec/cred.rowneruuid Resource to identify a key management service that supports this credential refresh method.
SKDC	oic.sec.crm.skdc	[1, 2, 4, 32]	The Server issues a request to obtain a ticket for the Device. The Server updates the credential using the information contained in the response to the ticket request. The Server uses its /oic/sec/cred.rowneruuid Resource to identify the key management service that supports this credential refresh method. The Server uses its /oic/sec/cred.rowneruuid Resource to identify a key management service that supports this credential refresh method.
PKCS10	oic.sec.crm.pk10	[8]	The Server issues a PKCS#10 certificate request message to obtain a new certificate. The Server uses its /oic/sec/cred.rowneruuid Resource to identify the key management service that supports this credential refresh method. The Server uses its /oic/sec/cred.rowneruuid Resource to identify a key management service that supports this credential refresh method.

Table 30 – Value Definition of the oic.sec.crmtype Property

13.2.1.1 Credential Usage

Credential Usage indicates to the Device the circumstances in which a credential should be used. Five values are defined:

- oic.sec.cred.trustca: This certificate is a trust anchor for the purposes of certificate chain validation, as defined in section 10.3.
- oic.sec.cred.cert: This credusage is used for certificates for which the Device possesses the private key and uses it for identity authentication in a secure session, as defined in section 10.3.

- oic.sec.cred.rolecert: This credusage is used for certificates for which the Device possesses the private key and uses to assert one or more roles, as defined in section 10.3.1.
- oic.sec.cred.mfgtrustca: This certificate is a trust anchor for the purposes of the Manufacturer Certificate Based Owner Transfer Method as defined in section 7.3.6.
- oic.sec.cred.mfgcert: This certificate is used for certificates for which the Device possesses the private key and uses it for authentication in the Manufacturer Certificate Based Owner Transfer Method as defined in section 7.3.6.

13.2.2 Key Formatting

13.2.2.1 Symmetric Key Formatting

Symmetric keys shall have the following format:

Name	Value	Type	Description
Length	16	OCTET	Specifies the number of 8-bit octets following Length
Key	opaque	OCTET Array	16 byte array of octets. When used as input to a PSK function Length is omitted.

Table 31 – 128-bit symmetric key

Name	Value	Type	Description
Length	32	OCTET	Specifies the number of 8-bit octets following Length
Key	opaque	OCTET Array	32 byte array of octets. When used as input to a PSK function Length is omitted.

Table 32 – 256-bit symmetric key

13.2.2.2 Asymmetric Keys

Note: Asymmetric key formatting is not available in this revision of the specification.

13.2.2.3 Asymmetric Keys with Certificate

Key formatting is defined by certificate definition.

13.2.2.4 Passwords

Technical Note: Password formatting is not available in this revision of the specification.

13.2.3 Credential Refresh Method Details

13.2.3.1.1 Provisioning Service

The resource owner identifies the provisioning service. If the Server determines a credential requires refresh and the other methods do not apply or fail, the Server will request re-provisioning of the credential before expiration. If the credential is allowed to expire, the Server should delete the Resource.

13.2.3.1.2 Pre-Shared Key

Using this mode, the current PSK is used to establish a Diffie-Hellmen session key in DTLS. The TLS_PRF is used as the key derivation function (KDF) that produces the new (refreshed) PSK.

PSK = TLS_PRF(MasterSecret, Message, length);

- MasterSecret – is the MasterSecret value resulting from the DTLS handshake using one of the above ciphersuites.
- Message is the concatenation of the following values:
  - RM - Refresh method – I.e. "oic.sec.crm.psk"
  - Device ID_A is the string representation of the Device ID that supplied the DTLS ClientHello.
  - Device ID_B is the Device responding to the DTLS ClientHello message
- Length of Message in bytes.

Both Server and Client use the PSK to update the /oic/sec/cred Resource's privatedata Property. If Server initiated the credential refresh, it selects the new validity period. The Server sends the chosen validity period to the Client over the newly established DTLS session so it can update it's corresponding credential Resource for the Server.

13.2.3.1.3 Random PIN

Using this mode, the current unexpired PIN is used to generate a PSK following RFC2898. The PSK is used during the Diffie-Hellman exchange to produce a new session key. The session key should be used to switch from PIN to PSK mode.

The PIN is randomly generated by the Server and communicated to the Client through an out-of-band method. The OOB method used is out-of-scope.

The pseudo-random function (PBKDF2) defined by RFC2898. PIN is a shared value used to generate a pre-shared key. The PIN-authenticated pre-shared key (PPSK) is supplied to a DTLS ciphersuite that accepts a PSK.

PPSK = PBKDF2(PRF, PIN, RM, Device ID, c, dkLen)

The PBKDF2 function has the following parameters:

- PRF – Uses the DTLS PRF.
- PIN – Shared between Devices.
- RM - Refresh method – I.e. "oic.sec.crm.rdp"
- Device ID – UUID of the new Device.
- c – Iteration count initialized to 1000, incremented upon each use.
- dkLen – Desired length of the derived PSK in octets.

Both Server and Client use the PPSK to update the /oic/sec/cred Resource's PrivateData Property. If Server initiated the credential refresh, it selects the new validity period. The Server sends the chosen validity period to the Client over the newly established DTLS session so it can update its corresponding credential Resource for the Server.

13.2.3.1.4 SKDC

A DTLS session is opened to the /oic/sec/cred.rowneruuid with svctype="oic.sec.cms" that supports the oic.sec.crm.skdc credential refresh method. A ticket request message is delivered to the oic.sec.cms service and in response returns the ticket request. The Server updates or instantiates an /oic/sec/cred Resource guided by the ticket response contents.

13.2.3.1.5 PKCS10

A DTLS session is opened to the /oic/sec/cred.rowneruuid with svctype="oic.sec.cms" that supports the oic.sec.crm.pk10 credential refresh method. A PKCS10 formatted message is delivered to the service. After the refreshed certificate is issued, the oic.sec.cms service pushes the certificate to the Server. The Server updates or instantiates an /oic/sec/cred Resource guided by the certificate contents.

13.2.3.2 Resource Owner

The Resource Owner Property allows credential provisioning to occur soon after Device onboarding before access to support services has been established. It identifies the entity authorized to manage the /oic/sec/cred Resource in response to Device recovery situations.

13.3 Certificate Revocation List

13.3.1 CRL Resource Definition

Device certificates and private keys are kept in `cred` Resource. CRL is maintained and updated with a separate `crl` Resource that is newly defined for maintaining the revocation list.

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/crl	CRLs	urn:oic.r.crl	baseline	Resource containing CRLs for Device certificate revocation	Security

Table 33 – Definition of the oic.r.crl Resource

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
CRL Id	crlid	UINT16	0 – 64K-1	RW	Yes	CRL ID for references from other Resource
This Update	thisupdate	String	-	RW	Yes	This indicates the time when this CRL has been updated.(UTC)
CRL Data	crldata	String	-	RW	Yes	CRL data based on CertificateList in CRL profile

Table 34 – Properties of the oic.r.crl Resource

13.4 ACL Resources

All Resource hosted by a Server are required to match an ACL policy. ACL policies can be expressed using three ACL Resource Types: /oic/sec/acl, /oic/sec/amacl and /oic/sec/sacl. The subject (e.g. Device ID of the Client) requesting access to a Resource shall be authenticated prior to applying the ACL check. Resources that are available to anyone can use a wildcard subject reference. All Resource accessible via the unsecured communication channel shall be named using the wildcard subject.

13.4.1 OCF Access Control List (ACL) BNF defines ACL structures.

ACL structure in Backus-Naur Form (BNF) notation:

`<ACL>`	`<ACE> {<ACE>}`
`<ACE>`	`<SubjectId> <ResourceRef> <Permission> {<Validity>}`
`<SubjectId>`	`<DeviceId> | <Wildcard> | <RoleId>`
`<DeviceId>`	`<UUID>`
`<RoleId>`	`[<Authority>] <RoleName> {<RoleName>}`
`<RoleName>`	`<URI>`
`<Authority>`	`<UUID>`
`<ResourceRef>`	`' (' <OIC_LINK> {',' {OIC_LINK>} ')'`
`<Permission>`	`('C' | '-') ('R' | '-') ('U' | '-') ('D' | '-') ('N' | '-')`
`<Validity>`	`<Period> {<Recurrence>}`

`<Wildcard>`	`'*'`
`<URI>`	`RFC3986 //` OCF Core Specification `defined`
`<UUID>`	`RFC4122 //` OCF Core Specification `defined`
`<Period>`	`RFC5545 Period`
`<Recurrence>`	`RFC5545 Recurrence`
`<OIC_LINK>`	OCF Core Specification `defined in JSON Schema`

Table 35 – BNF Definition of OCF ACL

The <DeviceId> token means the requestor must possess a credential that uses <UUID> as its identity in order to match the requestor to the <ACE> policy.

The <RoleID> token means the requestor must possess a role credential with <URI> as its role in order to match the requestor to the <ACE> policy.

The <Wildcard> token "*" means any requestor is matched to the <ACE> policy, with or without authentication.

When a <SubjectId> is matched to an <ACE> policy the <ResourceRef> is used to match the <ACE> policy to resources.

The <OIC_LINK> token contains values used to query existence of hosted resources.

The <Permission> token specifies the privilege granted by the <ACE> policy given the <SubjectId> and <ResourceRef> matching does not produce the empty set match.

Permissions are defined in terms of CREATE ('C'), RETRIEVE ('R'), UPDATE ('U'), DELETE ('D'), NOTIFY ('N') and NIL ('-'). NIL is substituted for a permissions character that signifies the respective permission is not granted.

The empty set match result defaults to a condition where no access rights are granted.

If the <Validity> token exists, the <Permission> granted is constrained to the time <Period>. <Validity> may further be segmented into a <Recurrence> pattern where access may alternatively be granted and rescinded according to the pattern.

13.4.2 ACL Resource

There are two types of ACLs, 'acl' is a list of type 'ace' and 'acl2' is a list of type 'ace2'. A Device shall not host the /acl Resource. Note: the /acl Resource is defined for backward compatibility and use by Provisioning Tools, etc.

In order to provide an interface which allows management of "aces2" (for /oic/sec/acl2 Resource) Array Property, the RETRIEVE, UPDATE and DELETE operations on the oic.r.ace2 Resource shall behave as follows:

1. A RETRIEVE shall return the full Resource representation.

2. An UPDATE shall replace or add to the Properties included in the representation sent with the UPDATE request, as follows:

    a. If an UPDATE representation includes the array Property, then:

        i. Supplied ACEs with an "aceid" that matches an existing "aceid" shall replace completely the corresponding ACE in the existing "aces2" array.

        ii. Supplied ACEs without an "aceid" shall be appended to the existing "aces2" array, and a unique (to the acl2 Resource) "aceid" shall be created and assigned to the new ACE by the Server. The "aceid" of a deleted ACE should not be reused, to improve the determinism of the interface and reduce opportunity for race conditions.

iii. Supplied ACEs with an "aceid" that does not match an existing "aceid" shall be appended to the existing "aces2" array, using the supplied "aceid".

3. A DELETE without query parameters shall remove the entire "aces2" array, but shall not remove the oic.r.ace2 Resource.

4. A DELETE with one or more "aceid" query parameters shall remove the ACE(s) with the corresponding aceid(s) from the "aces2" array.

Evaluation of local ACL Resource completes when all ACL Resource have been queried and no entry can be found for the requested Resource for the requestor – e.g. /oic/sec/acl, /oic/sec/sacl and /oic/sec/amacl do not match the subject and the requested Resource.

If an access manager ACL satisfies the request, the Server opens a secure connection to the AMS. If the primary AMS is unavailable, a secondary AMS should be tried. The Server queries the AMS supplying the subject and requested resource as filter criteria. The Server Device ID is taken from the secure connection context and included as filter criteria by the AMS. If the AMS policy satisfies the Permission Property is returned.

If the requested Resource is still not matched, the Server returns an error. The requester should query the Server to discover the configured AMS services. The Client should contact the AMS to request a sacl (/oic/sec/sacl) Resource. Performing the following operations implement this type of request:

1. Client: Open secure connection to AMS.
2. Client: GET /oic/sec/acl?device="urn:uuid:XXX...",resource="URI"
3. AMS: constructs a /oic/sec/sacl Resource that is signed by the AMS and returns it in response to the GET command.
4. Client: POST /oic/sec/sacl [{ ...sacl... }]
5. Server: verifies sacl signature using AMS credentials and installs the ACL Resource if valid.
6. Client: retries original Resource access request. This time the new ACL is included in the local acl evaluation.

The ACL contained in the /oic/sec/sacl Resource should grant longer term access that satisfies repeated Resource requests.

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/acl	ACL	urn:oic.r.acl	baseline	Resource for managing access	Security

Table 36 – Definition of the oic.r.acl Resource

Property Title	Property Name	Value Type	Value Rule	Mandatory	Access Mode	Device State	Description
ACE List	aclist	oic.sec.ace	-	Yes		-	Access Control Entries in the ACL resource. This Property contains "aces", an array of oic.sec.ace1 resources and "aces2", an array of oic.sec.ace2 Resources
					R	RESET	Server shall set to manufacturer defaults.
					RW	RFOTM	The OBT shall configure select oic.sec.ace2 entries after a secure session is established.
					RW	RFPRO	The AMS (referenced via rowneruuid property) shall update the oic.sec.ace2 entries after mutually authenticated secure session is established. Access to vertical resources is prohibited.
					R	RFNOP	Access to vertical resources is permitted after a matching ACE is found.
					RW	SRESET	The OBT (referenced via devowneruuid property) should evaluate the integrity of and may update oic.sec.ace2 entries when a secure session is established and the Server and OBT are authenticated.
Resource Owner ID	rowneruuid	String	uuid	Yes	-	-	The resource owner property (rowneruuid) is used by the Server to reference a service provider trusted by the Server. Server shall verify the service provider is authorized to perform the requested action
					R	RESET	Server shall set to the nil uuid value (e.g. "00000000-0000-0000-0000-000000000000")
					RW	RFOTM	The DOXS should configure the /acl.rowneruuid and /acl2.rowneruuid property when a successful owner transfer session is established.
					R	RFPRO	n/a
					R	RFNOP	n/a
					RW	SRESET	The DOXS (referenced via /doxm.devowneruuid property) should verify and if needed, update the resource owner property when a mutually authenticated secure session is established. If the rowneruuid does not refer to a valid DOXS the Server shall transition to RESET Device state..

Table 37 – Properties of the oic.r.acl Resource

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
Resources	resources	oic.oic-link	array		Yes	The application's Resources to which a security policy applies
Permission	permission	oic.sec.crudntype	bitmask	RW	Yes	Bitmask encoding of CRUDN permission
Validity	validity	oic.sec.ace/definitions/time-interval	array	RW	No	An array of a tuple of period and recurrence. Each item in this array contains a string representing a period using the RFC5545 Period, and a string array representing a recurrence rule using the RFC5545 Recurrence.
Subject ID	subjectuuid	String	uuid, "*"	RW	Yes	A uuid that identifies the Device to which this ACE applies to or "*" for anonymous access.

Table 38 – Properties of the oic.r.ace Property

Value	Access Policy	Description	Notes
bx0000,0000 (0)	No permissions	No permissions	
bx0000,0001 (1)	C	CREATE	
bx0000,0010 (2)	R	RETREIVE, OBSERVE, DISCOVER	Note that the "R" permission bit covers both the Read permission and the Observe permission.
bx0000,0100 (4)	U	WRITE, UPDATE	
bx0000,1000 (8)	D	DELETE	
bx0001,0000 (16)	N	NOTIFY	The "N" permission bit is ignored in OCF 1.0, since "R" covers the Observe permission. It is documented for future versions

Table 39 – Value Definition of the oic.sec.crudntype Property

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/acl2	ACL2	oic.r.acl2	baseline	Resource for managing access	Security

Table 40 – Definition of the oic.sec.acl2 Resource

Property Name	Value Type	Mandatory	Device State	Access Mode	Description
aclist2	array of oic.sec.ace2				The aclist2 property is an array of ACE records of type "oic.sec.ace2". The Server uses this list to apply access control to its local resources.
			RESET	R	Server shall set to manufacturer defaults.
			RFOTM	RW	The OBT shall configure select oic.sec.ace2 entries after a secure session is established.
			RFPRO	RW	The AMS (referenced via rowneruuid property) shall update the oic.sec.ace2 entries after mutually authenticated secure session is established. Access to vertical resources is prohibited.
			RFNOP	R	Access to vertical resources is permitted after a matching ACE is found.
			SRESET	RW	The OBT (referenced via devowneruuid property) should evaluate the integrity of and may update oic.sec.ace2 entries when a secure session is established and the Server and OBT are authenticated.
rowneruuid	uuid	Yes			Same as rowneruuid in oic.sec.acl

Table 41 – Properties of the oic.sec.acl2 Resource

Property Name	Value Type	Mandatory	Description
subject	oic.sec.roletype, oic.sec.didtype, oic.sec.conntype	Yes	The Client is the subject of the ACE when the roles, Device ID, or connection type matches.
resources	array of oic.sec.ace2.resource-ref	Yes	The application's resources to which a security policy applies
permission	oic.sec.crudntype.bitmask	Yes	Bitmask encoding of CRUDN permission
validity	array of oic.sec.time-pattern	No	An array of a tuple of period and recurrence. Each item in this array contains a string representing a period using the RFC5545 Period, and a string array representing a recurrence rule using the RFC5545 Recurrence.
aceid	integer	Yes	An aceid is unique with respect to the 'aces' array.

Table 42 – oic.sec.ace2 data type definition.

Property Name	Value Type	Mandatory	Description
href	uri	No	A URI referring to a resource to which the containing ACE applies
rt	array of strings	No	The resource types to which the containing ACE applies
if	array of strings	No	The interfaces to which the containing ACE applies
wc	string	No	A wildcard matching policy where: "+" – Matches all discoverable resources "-" – Matches all non-discoverable resources "*" – Matches all resources

Table 43 – oic.sec.ace2.resource-ref data type definition.

Property Name	Value Type	Value Rule	Description
conntype	string	enum ["auth-crypt", "anon-clear"]	This property allows an ACE to be matched based on the connection or message protection type
		auth-crypt	ACE applies if the Client is authenticated and the data channel or message is encrypted and integrity protected
		anon-clear	ACE applies if the Client is not authenticated and the data channel or message is not encrypted but may be integrity protected

Table 44 – Value definition oic.sec.conntype Property

Local ACL Resources supply policy to a Resource access enforcement point within an OCF stack instance. The OCF framework gates Client access to Server Resources. It evaluates the subject's request using policy in the ACL.

Resources named in the ACL policy should be fully qualified or partially qualified. Fully qualified Resource references should include the Device ID of a remote Device hosting the Resources.

Partially qualified references means the local Resource Server is hosting the Resource. If a fully qualified resource reference is given, the Intermediary enforcing access shall have a secure channel to the Resource Server and the Resource Server shall verify the Intermediary is authorized to act on its behalf as a Resource access enforcement point.

Resource Servers should include references to Device and ACL Resources where access enforcement is to be applied. However, access enforcement logic shall not depend on these references for access control processing as access to Server Resources will have already been granted.

Local ACL Resources identify a Resource Owner service that is authorized to instantiate and modify this Resource. This prevents non-terminating dependency on some other ACL Resource. Nevertheless, it should be desirable to grant access rights to ACL Resources using an ACL Resource.

An ACE or ACE2 entry is called *currently valid* if the validity period of the ACE or ACE2 entry includes the time of the request. Note that the validity period in the ACE or ACE2 may be a recurring time period (e.g., daily from 1:00-2:00). Matching the resource(s) specified in a request to the resource property of the ACE or ACE2 is defined in Section 12.2. For example, one way they can match is if the Resource URI in the request exactly matches one of the resource references in the ACE or ACE2 entries.

A request will match an ACE if any of the following are true:

1. The deviceuuid associated with the secure session matches the "subjectuuid" of the ACE; AND the resource of the request matches one of the "resources" of the ACE; AND the ACE is currently valid.

2. The ACE "subjectuuid" contains the wildcard "*" character; AND the resource of the request matches one of the "resources" of the ACE; AND the ACE is currently valid.

3. When authentication uses a symmetric key credential;

   AND the CoAP payload query string of the request specifies a role, which is associated with the symmetric key credential of the current secure session;

   AND the CoAP payload query string of the request specifies a role, which is contained in the oic.r.cred.creds.roleid property of the current secure session;

   AND the resource of the request matches one of the "resources" of the ACE;

   AND the ACE is currently valid.

A request will match an ACE2 if any of the following are true:

1. The ACE2 "subject" is of type oic.sec.didtype has a UUID value that matches the deviceuuid associated with the secure session;

   AND the resource of the request matches one of the "resources" of the ACE2 oic.sec.ace2.resource-ref;

   AND the ACE2 is currently valid.

2. The ACE2 "subject" is of type oic.sec.conntype and has the wildcard value that matches the currently established connection type;

   AND the resource of the request matches one of the "resources" of the ACE2 oic.sec.ace2.resource-ref;

AND the ACE2 is currently valid.

3. When Client authentication uses a certificate credential;

   AND one of the roleid values contained in the role certificate matches the "roleid" of the ACE2 oic.sec.roletype;

   AND the role certificate public key matches the public key of the certificate used to establish the current secure session;

   AND the resource of the request matches one of the "resources" of the ACE2 oic.sec.ace2.resource-ref;

   AND the ACE2 is currently valid.

4. When Client authentication uses a certificate credential;

   AND the CoAP payload query string of the request specifies a role, which is member of the set of roles contained in the role certificate;

   AND the roleid values contained in the role certificate matches the "roleid" of the ACE2 oic.sec.roletype;

   AND the role certificate public key matches the public key of the certificate used to establish the current secure session;

   AND the resource of the request matches one of the "resources" of the ACE2 oic.sec.ace2.resource-ref;

   AND the ACE2 is currently valid.

5. When Client authentication uses a symmetric key credential;

   AND one of the roleid values associated with the symmetric key credential used in the secure session, matches the "roleid" of the ACE2 oic.sec.roletype;

   AND the resource of the request matches one of the "resources" of the ACE2 oic.sec.ace2.resource-ref;

   AND the ACE2 is currently valid.

6. When Client authentication uses a symmetric key credential;

   AND the CoAP payload query string of the request specifies a role, which is contained in the oic.r.cred.creds.roleid property of the current secure session;

   AND CoAP payload query string of the request specifies a role that matches the "roleid" of the ACE2 oic.sec.roletype;

   AND the resource of the request matches one of the "resources" of the ACE2 oic.sec.ace2.resource-ref;

   AND the ACE2 is currently valid.

A request is granted if ANY of the 'matching' ACEs contains the permission to allow the request. Otherwise, the request is denied.

Note that there is no way for an ACE to explicitly deny permission to a resource. Therefore, if one Device with a given role should have slightly different permissions than another Device with the same role, they must be provisioned with different roles.

13.5 Access Manager ACL Resource

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/amacl	Managed ACL	urn:oic.r.amacl	baseline	Resource for managing access	Security

Table 45 – Definition of the oic.r.amacl Resource

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
Resources	resources	oic.sec.ace2	array	RW	Yes	Multiple links to this host's Resources

Table 46 – Properties of the oic.r.amacl Resource

13.6 Signed ACL Resource

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/sacl	Signed ACL	urn:oic.r.sacl	baseline	Resource for managing access	Security

Table 47 – Definition of the oic.r.sacl Resource

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
ACE List	aclist2	oic.sec.ace2	array	RW	Yes	Access Control Entries in the ACL Resource
Signature	signature	oic.sec.sigtype	-	RW	Yes	The signature over the ACL Resource

Table 48 – Properties of the oic.r.sacl Resource

Property Title	Property Name	Value Type	Value Rule	Unit	Access Mode	Mandatory	Description
Signature Type	sigtype	String	-	-	RW	Yes	The string specifying the predefined signature format. "oic.sec.sigtype.jws" – RFC7515 JSON web signature (JWS) object "oic.sec.sigtype.pk7" – RFC2315 base64-encoded object "oic.sec.sigtype.cws" – CBOR-encoded JWS object
Signature Value	sigvalue	String	-	-	RW	Yes	The encoded signature

Table 49 – Properties of the oic.sec.sigtype Property

13.7 Provisioning Status Resource

The **/oic/sec/pstat** Resource maintains the Device provisioning status. Device provisioning should be Client-directed or Server-directed. Client-directed provisioning relies on a Client Device to determine what, how and when Server Resources should be instantiated and updated. Server-directed provisioning relies on the Server to seek provisioning when conditions dictate. Server-directed provisioning depends on configuration of the /oic/sec/cred.rowneruuid and /oic/sec/cred Resources, at least minimally, to bootstrap the Server with settings necessary to open a secure connection with appropriate support services.

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/pstat	Provisioning Status	urn:oic.r.pstat	baseline	Resource for managing Device provisioning status	Configuration

Table 50 – Definition of the oic.r.pstat Resource

Property Title	Property Name	Value Type	Value Rule	Mandatory	Access Mode	Device State	Description
Device Onboarding State	dos	oic.sec.dostype	-	Yes	RW		Device Onboarding State
Is Operational	isop	Boolean	T|F	Yes	R	RESET	Device can function even when Cm is non-zero. Device will only service requests related to satisfying Tm when IsOp is FALSE. Server shall set to FALSE
					R	RFOTM	Server shall set to FALSE
					R	RFPRO	Server shall set to FALSE
					R	RFNOP	Server shall set to TRUE
					R	SRESET	Server shall set to FALSE
Current Mode	cm	oic.sec.dpmtype	bitmask	Yes	R		Server shall set to 0000,0001
					R		Should be set by DOXS after successful OTM to 00xx,xx10.
					R		Set by CMS, AMS, DOXS after successful authentication
					R		Set by CMS, AMS, DOXS after successful authentication
					R		Server shall set to 0000,0001
Target Mode	tm	oic.sec.dpmtype	bitmask	No	R		Server shall set to 0000,0010
					RW		Set by DOXS after successful OTM
					RW		Set by CMS, AMS, DOXS after successful authentication
					RW		Set by CMS, AMS, DOXS after successful authentication
					RW		Set by DOXS as needed to recover from failures. Server shall set to XXXX,XX00 upon entry into SRESET.
Operational Mode	om	oic.sec.pomtype	bitmask	Yes	R		Server shall set to manufacturer default.
					RW		Set by DOXS after successful OTM
					RW		Set by CMS, AMS, DOXS after successful authentication
					RW		Set by CMS, AMS, DOXS after successful authentication
					RW		Set by DOXS.
Supported Mode	sm	oic.sec.pomtype	bitmask	Yes	R		Supported provisioning services operation modes

Device UUID	deviceuuid	String	uuid	Yes	RW		[DEPRECATED] A uuid that identifies the Device to which the status applies
Resource Owner ID	rowneruuid	String	uuid	Yes	R	RESET	Server shall set to the nil uuid value (e.g. "00000000-0000-0000-0000-000000000000")
					RW	RFOTM	The DOXS should configure the /pstat.rowneruuid property when a successful owner transfer session is established.
					R	RFPRO	n/a
					R	RFNOP	n/a
					RW	SRESET	The DOXS (referenced via /doxm.devowneruuid property) should verify and if needed, update the resource owner property when a mutually authenticated secure session is established. If the rowneruuid does not refer to a valid DOXS the Server shall transition to RESET Device state.

Table 51 – Properties of the oic.r.pstat Resource

The provisioning status Resource /oic/sec/pstat is used to enable Devices to perform self-directed provisioning. Devices are aware of their current configuration status and a target configuration objective. When there is a difference between current and target status, the Device should consult the rowneruuid Property of /oic/sec/cred Resource to discover whether any suitable provisioning services exist. The Device should request provisioning if configured to do so. The om Property of /oic/sec/pstat Resource will specify expected Device behaviour under these circumstances.

Self-directed provisioning enables Devices to function with greater autonomy to minimize dependence on a central provisioning authority that should be a single point of failure in the network.

Property Title	Property Name	Value Type	Value Rule	Mandatory	Access Mode	Device State	Description
Device Onboarding State	s	UINT16	enum (0=RESET, 1=RFOTM, 2=RFPRO, 3=RFNOP, 4=SRESET	Y	R	RESET	The Device is in a hard reset state.
					RW	RFOTM	The Device is in a Ready-For-Owner-Transfer-Method state. Set by DOXS after successful OTM to RFPRO.
					RW	RFPRO	The Device is in a Ready-For-PROvisioning state. Set by CMS, AMS, DOXS after successful authentication
					RW	RFNOP	The Device is in a Ready-For-Normal-Operation state. Set by CMS, AMS, DOXS after successful authentication
					RW	SRESET	The Device is in a Soft-Reset state. State transitions can be effected remotely by writing to /pstat.dos.s when the caller is authenticated and authorized to update the dos.s property. Set by CMS, AMS, DOXS after successful authentication
Pending state	p	Boolean	T | F	Y	R		TRUE (1) – 's' state is pending until all necessary changes to Device resources are complete FALSE (0) – 's' state changes are complete

Table 52 – Properties of the oic.sec.dostype Property

In all states:

- The /pstat.dos.p Property is read-only by all requestors.
- An authenticated client can effect a Device state change by updating pstat.dos.s=<device_state>. Doing so instructs the Server to automatically perform all the changes required to transition to the designated Device state. There may be multiple steps, hence the dos.p value is set to TRUE as the first step and remains TRUE until all steps are complete. The final step sets dos.p to FALSE. The dos.s value is set to the designated Device state at the same time the dos.p value is set to FALSE. A client may observe /pstat.dos to be notified when a Device state change is completed.
- The Client is authenticated to write to /pstat.dos.s if it possesses an appropriate role (e.g. DOXS, CMS, AMS).
- Write requests to /pstat.dos.s where the requestor tries to transition to a state that isn't reachable will result in a DEVICE_STATE_NOT_PERMITTED error.
- Write requests to /pstat.dos.s when /pstat.dos.p is TRUE will result in a DEVICE_STATE_NOT_READY error.
- /pstat.dos.p must be set to TRUE on entrance.
- /pstat.dos.p property must be set to FALSE on exit.

When Device state is RESET:

- All SVR content is removed and reset to manufacturer default values.
- The default manufacturer Device state is RESET.
- Vertical resources are reset to manufacturer default values.
- Vertical resources are inaccessible.
- If client subscribers OBSERVE dos.p=FALSE, the notification is sent prior to the point where access control and credential resources (needed to deliver the message) are dismantled.
- After successfully processing RESET the SRM transitions to RFOTM by setting /pstat.dos.s to RFOTM.

When Device state is RFOTM:

- Vertical Resources are inaccessible.
- Before OTM is successful, the deviceid Property of /oic/sec/doxm Resource must be set to a randomized UUID value.
- Before OTM is successful, the /pstat.dos.s Property is read-only by unauthenticated requestors
- After the OTM is successful, the /pstat.dos.s Property is read-write by authorized requestors.
- The negotiated Device owner credential is used to create an authenticated session over which the OBT directs the Device state to transition to RFPRO.
- If an authenticated session cannot be established when the OTM completes after <tbd=60> seconds, the SRM asserts the OTM failed and transitions to RESET (/pstat.dos.s=RESET). (Note: The transfer of ownership is considered complete when /doxm.owned is set to TRUE. The Device state may continue in RFOTM to complete initial provisioning.)

When Device state is RFPRO:

- The /pstat.dos.s Property is read-only by unauthorized requestors and read-write by authorized requestors.
- Vertical Resources are inaccessible.
- The OCF Server may re-create vertical Resources.
- An authorized Client may provision SVRs as needed for normal functioning in RFNOP.
- An authorized Client may perform consistency checks on SVRs to determine which shall be re-provisioned.
- Failure to successfully provision SVRs may trigger a state change to RESET. For example, if the Device has already transitioned from SRESET but consistency checks continue to fail.
- The authorized Client sets the /pstat.dos.s=RFNOP.

When Device state is RFNOP:

- The /pstat.dos.s Property is read-only by unauthorized requestors and read-write by authorized requestors.
- Vertical resources, SVRs and core Resources are accessible following normal access processing.
- An authorized may transition to RFPRO. Only the Device owner may transition to SRESET or RESET.

When Device state is SRESET:

- Vertical Resources are inaccessible. The integrity of vertical Resources may be suspect but the SRM doesn’t attempt to access or reference them.
- SVR integrity is not guaranteed, but access to some SVR Properties is necessary. These include /doxm.devowner, /cred[<devowner>] and /pstat.dos.
- The certificates that identify and authorize the Device owner are sufficient to re-create minimalist /cred and /doxm resources enabling Device owner control of SRESET. If the SRM can’t establish these Resources, then it will transition to RESET state.
- An authorized Client performs SVR consistency checks. The caller may provision SVRs as needed to ensure they are available for continued provisioning in RFPRO or for normal functioning in RFNOP.
- The authorized Device owner may avoid entering RESET state and RFOTM by writing RFPRO or RFNOP to /pstat.dos.s.
- ACLs on SVR are presumed to be invalid. Access authorization is granted according to Device owner privileges.
- The SRM asserts a Client-directed operational mode (e.g. /pstat.om=CLIENT_DIRECTED).

The *provisioning mode* type is a 16-bit mask enumerating the various Device provisioning modes. "{ProvisioningMode}" should be used in this document to refer to an instance of a provisioning mode without selecting any particular value.

Type Name	Type URN	Description
Device Provisioning Mode	urn:oic.sec.dpmtype	Device provisioning mode is a 16-bit bitmask describing various provisioning modes

Table 53 – Definition of the oic.sec.dpmtype Property

Value	Device Mode	Description
bx0000,0001 (1)	Reset	Device reset mode enabling manufacturer reset operations
bx0000,0010 (2)	Take Owner	Device pairing mode enabling owner transfer operations
bx0000,0100 (4)	Bootstrap Service	Service provisioning mode enabling instantiation of a BSS. This allows authorized entities to install a BSS.
bx0000,1000 (8)	Security Management Services	Service provisioning mode enabling instantiation of Device security services and related credentials
bx0001,0000 (16)	Provision Credentials	Credential provisioning mode enabling instantiation of pairwise Device credentials using a management service of type urn:oic.sec.cms
bx0010,0000 (32)	Provision ACLs	ACL provisioning mode enabling instantiation of Device ACLs using a management service of type urn:oic.sec. ams
bx0100,0000 (64)	Initiate Software Version Validation	Software version validation requested/pending (1) Software version validation complete (0)
bx1000,0000 (128)	Initiate Secure Software Update	Secure software update requested/pending (1) Secure software update complete (0)

Table 54 – Value Definition of the oic.sec.dpmtype Property (Low-Byte)

Value	Device Mode	Description
bx0000,0000 – bx1111,1111	<Reserved>	Reserved for later use

Table 55 – Value Definition of the oic.sec.dpmtype Property (High-Byte)

The *provisioning operation mode* type is a 8-bit mask enumerating the various provisioning operation modes.

Type Name	Type URN	Description
Device Provisioning OperationMode	urn:oic.sec.pomtype	Device provisioning operation mode is a 8-bit bitmask describing various provisioning operation modes

Table 56 – Definition of the oic.sec.pomtype Property

Value	Operation Mode	Description
bx0000,0001 (1)	Server-directed utilizing multiple provisioning services	Provisioning related services are placed in different Devices. Hence, a provisioned Device should establish multiple DTLS sessions for each service. This condition exists when bit 0 is FALSE.
bx0000,0010 (2)	Server-directed utilizing a single provisioning service	All provisioning related services are in the same Device. Hence, instead of establishing multiple DTLS sessions with provisioning services, a provisioned Device establishes only one DTLS session with the Device. This condition exists when bit 0 is TRUE.
bx0000,0100 (4)	Client-directed provisioning	Device supports provisioning service control of this Device's provisioning operations. This condition exists when bit 1 is TRUE. When this bit is FALSE this Device controls provisioning steps.
bx0000,1000(8) – bx1000,0000(128)	<Reserved>	Reserved for later use
bx1111,11xx	<Reserved>	Reserved for later use

Table 57 – Value Definition of the oic.sec.pomtype Property

13.8 Certificate Signing Request Resource

The /oic/sec/csr Resource is used by a Device to provide its desired identity, public key to be certified, and a proof of possession of the corresponding private key in the form of a RFC 2986 PKCS#10 Certification Request. If the Device supports certificates (the sct property of /oic/sec/doxm has a 1 in the 0x8 bit position), the Device shall have a /oic/sec/csr resource.

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/csr	Certificate Signing Request	urn:oic.r.csr	baseline	The CSR resource contains a Certificate Signing Request for the Device's public key.	Configuration

Table 58 – Definition of the oic.r.csr Resource

Property Title	Property Name	Value Type	Access Mode	Mandatory	Description
Certificate Signing Request	csr	String	R	Yes	Contains the signed CSR encoded according to the encoding Property
Encoding	encoding	String	R	Yes	A string specifying the encoding format of the data contained in the csr Property "oic.sec.encoding.pem" – Encoding for PEM-encoded certificate signing request "oic.sec.encoding.der" – Encoding for DER-encoded certificate signing request

Table 59 – Properties of the oic.r.csr Resource

The Device chooses which public key to use, and may optionally generate a new key pair for this purpose.

In the CSR, the Common Name component of the Subject Name shall contain a string of the format "uuid:X" where X is the Device's requested UUID in the format defined by RFC 4122. The Common Name, and other components of the Subject Name, may contain other data. If the Device chooses to include additional information in the Common Name component, it shall delimit it from the UUID field by white space, a comma, or a semicolon.

If the Device does not have a pre-provisioned key pair to use, but is capable and willing to generate a new key pair, the Device may begin generation of a key pair as a result of a RETRIEVE of this resource. If the Device cannot immediately respond to the RETRIEVE request due to time required to generate a key pair, the Device shall return an "operation pending" error. This indicates to the Client that the Device is not yet ready to respond, but will be able at a later time. The Client should retry the request after a short delay.

13.9 Roles resource

The roles resource maintains roles that have been asserted with role certificates, as described in Section 10.3.1. Asserted roles have an associated public key, i.e., the public key in the role certificate. Clients may only access roles associated with their public key of the certificate used to authenticate during (D)TLS session establishment. The roles resource should be viewed as an extension of the (D)TLS session state. See section 10.3.1 for how role certificates are validated.

The roles resource shall be created by the server upon establishment of a secure (D)TLS session with a client, if is not already created. A server shall retain the roles resource at least as long as the (D)TLS session exists. A server shall retain each certificate in the roles resource at least until the certificate expires or the (D)TLS session ends, whichever is sooner. A server may retain the roles resource and its contents beyond the length of a (D)TLS session or a certificate's validity period, although the requirements of section 10.3 and 10.3.1 to validate a certificate's time validity at the point of use always apply. A server should regularly inspect the contents of the roles resource and purge contents based on a policy it determines based on its resource constraints.

For example, expired certificates, and certificates from clients that have not been heard from for some arbitrary period of time could be candidates for purging.

As stated above, the resource is implicitly created by the server upon establishment of a (D)TLS session. In more detail, the RETRIEVE, UPDATE and DELETE operations on the Roles Resource should behave as follows. Unlisted operations are implementation specific and not reliable. Note that this description is editorial, and the RAML provides the normative and formal behaviour description.

1. Retrieve shall return all previously asserted roles associated with the client's public key. Note that the public key is always available to the server as part of the secure channel information. Retrieve with query parameters is not supported.

2. Update includes the "roles" array property and distinct roles in this array are added to the resource. This is also scoped to the client's public key. Two roles are distinct if either of the "role" or "authority" properties differs.

3. Delete shall remove the entire "roles" array for the client's public key.

Fixed URI	Resource Type Title	Resource Type ID ("rt" value)	Interfaces	Description	Related Functional Interaction
/oic/sec/roles	Roles	urn:oic.r.roles	baseline	Resource containing roles that have previously been asserted to this server	Security

Table 60 – Definition of the oic.r.roles Resource

Property Title	Property Name	Value Type	Value Rule	Access Mode	Mandatory	Description
Roles	roles	oic.sec.cred	array	RW	Yes	List of roles previously asserted to this server

Table 61 – Properties of the oic.r.roles Resource

13.10 Security Virtual Resources (SVRs) and Access Policy

The SVRs expose the security-related Properties of the Device.

Granting access requests (RETRIEVE, UPDATE, DELETE, etc.) for these SVRs to unauthenticated (anonymous) Clients could create privacy or security concerns.

For example, when the Device onboarding State is RFOTM, it is necessary to grant requests for the oic.r.doxm Resource to anonymous requesters, so that the Device can be discovered and onboarded by an OBT. Subsequently, it might be preferable to deny requests for the oic.r.doxm Resource to anonymous requesters, to preserve privacy.

13.11 SVRs, Discoverability and Endpoints

All implemented SVRs shall be "discoverable" (reference OCF Core Specification, Policy Parameter section 7.8.2.1.2).

All implemented discoverable SVRs shall expose a Secure Endpoint (e.g. CoAPS) (reference OCF Core Specification, Endpoint chapter 10).

The /doxm Resource shall expose an Unsecure Endpoint (e.g. CoAP) in RFOTM (reference OCF Core Specification, Endpoint chapter 10).

13.12 Privacy Consideration for Core and SVRs

Unique identifiers are a privacy consideration due to their potential for being used as a tracking mechanism. These include the following Resources and Properties:

- /d Resource containing the ‘di’ and ‘piid’ Properties.
- /p Resource containing the ‘pi’ Property.
- /doxm Resource containing the ‘deviceuuid’ Property.

All identifiers are unique values that are visible to throughout the Device lifecycle by anonymous requestors. This implies any Client Device, including those with malicious intent, are able to reliably obtain identifiers useful for building a log of activity correlated with a specific Platform and Device.

There are two strategies for privacy protection of Devices:

1. Apply an ACL policy that restricts read access to Resources containing unique identifiers
2. Limit identifier persistence to make it impractical for tracking use.

Both techniques can be used effectively together to limit exposure to privacy attacks.

1. A Platform / Device manufacturer should specify a default ACL policy that restricts anonymous requestors from accessing unique identifiers. A network administrator should modify the ACL policy to grant access to authenticated Devices who, presumably, do not present a privacy threat.
2. Servers shall supply a temporary, non-repeating Device ID when the ‘owned’ Property in the /doxm Resource is FALSE and applies over multiple ownership transitions. The temporary identifiers are disjoint from and not correlated to the persistent identifiers shall be:
    a. Disjoint from (i.e. not linked) the persistent identifiers
3. Generated by a function that is pre-image resistant, second pre-image resistant and collision resistant

A new Device seeking deployment needs to inform would-be OBTs of the identifier used to begin the onboarding process. However, attackers could obtain the value too and use it for Device tracking throughout the Device’s lifetime. To address this privacy threat, Servers shall supply the temporary ‘deviceuuid’ to unauthenticate /oic/res requests when the 'deviceowneruuid' is the nil UUID. The Server shall generate a new pseudo-random temporary ‘deviceuuid’ value when the Device state transitions to RESET. This ensures the ‘deviceuuid’ value cannot be used to track across multiple owners.

The 'deviceowneruuid’ property is initialized to the nil UUID at RESET which is retained until being set during RFOTM Device state. The Device shall supply a non-persistent identifier to RETRIEVE requests on /oic/sec/doxm and /oic/res Resources while ‘deviceowneruuid’ is the nil UUID. The onboarding utility shall update the ‘devowneruuid’ to a non-nil UUID value that triggers the Server to allow the persistent ‘deviceuuid’ to be returned in RETRIEVE requests to the /doxm and /res resources.

The onboarding utility may also provision an ACL policy that restricts access to the /oic/sec/doxm resource such that only authenticated Clients are able to obtain the persistent ‘deviceuuid’ value. Clients avoid making unauthenticated discovery requests by having been provisioned with a /oic/sec/cred resource entry that contains the Server’s ‘deviceuuid’.

The ‘di’ property in the /oic/d Resource shall mirror that of the ‘deviceuuid’ Property. The onboarding utility should provision an ACL policy that restricts access to the /oic/d Resource such that only authenticated Clients are able to obtain the persistent ‘di’ value.

The ‘piid’ Property in the /oic/d Resource similarly should present a temporary and changing value when ‘deviceowneruuid’ has the nil UUID value. The server shall provide a persistent value (or allows the onboarding utility to provision) subsequent to ‘deviceowneruuid’ being changed to a non-nil UUID. An ACL policy on the /d resource protects the ‘piid’ from being disclosed to anonymous requestors.

The ‘pi’ Property in the /oic/p Resource shall have a temporary and changing value when ‘deviceowneruuid’ has the nil UUID value and shall change to a persistent value upon ‘deviceowneruuid’ being changed to a non-nil UUID. An ACL policy shall protect the ‘pi’ property in the /p resource from being disclosed to anonymous requestors.

Resource Type	**Property title**	**Property name**	**Value type**	**Access Mode**		**Behavior**
oic.wk.p	Platform ID	pi	oic.types-schema.uuid	All States	R	Server shall construct a temporary random UUID. (does not override the persistent pi)
oic.wk.p	Protocol Independent Identifier	piid	oic.types-schema.uuid	RESET, SRESET RFPRO, RFNOP	R	Server should construct a temporary random UUID when entering RESET state.
				RFOTM	RW	DOXS may set the persistent value after secure owner transfer session is established; otherwise Server sets value.
oic.wk.d	Device Identifier	di	oic.types-schema.uuid	All states	R	/d.di shall mirror the value contained in /doxm.deviceuuid in all Device states.

Table 62 – Core Resource Properties state

14 Core Interaction Patterns Security

This section is left intentionally blank and will be defined in future version..

14.1 Observer

14.2 Subscription/Notification

14.3 Groups

14.4 Publish-subscribe Patterns and Notification

15 Security Hardening Guidelines/ Execution Environment Security

This is an informative section. Many TGs in OCF have security considerations for their protocols and environments. These security considerations are addressed through security mechanisms specified in the security specifications for OCF. However, effectiveness of these mechanisms depends on security robustness of the underlying hardware and software Platform. This section defines the components required for execution environment security.

15.1 Execution environment elements

Execution environment within a computing Device has many components. To perform security functions in a robustness manner, each of these components has to be secured as a separate dimension. For instance, an execution environment performing AES cannot be considered secure if the input path entering keys into the execution engine is not secured, even though the partitions of the CPU, performing the AES encryption, operate in isolation from other processes. Different dimensions referred to as elements of the execution environment are listed below. To qualify as a secure execution environment (SEE), the corresponding SEE element must qualify as secure.

- (Secure) Storage
- (Secure) Execution engine
- (Trusted) Input/output paths
- (Secure) Time Source/clock
- (Random) number generator
- (Approved) cryptographic algorithms
- Hardware Tamper (protection)

Note that software security practices (such as those covered by OWASP) are outside scope of this specification, as development of secure code is a practice to be followed by the open source development community. This specification will however address the underlying Platform assistance required for executing software. Examples are secure boot and secure software upgrade.

Each of the elements above are described in the following subsections.

15.1.1 Secure Storage

Secure storage refers to the physical method of housing sensitive or confidential data ("Sensitive Data"). Such data could include but not be limited to symmetric or asymmetric private keys, certificate data, network access credentials, or personal user information. Sensitive Data requires that its integrity be maintained, whereas *Critical* Sensitive Data requires that both its integrity and confidentiality be maintained.

It is strongly recommended that IoT Device makers provide reasonable protection for Sensitive Data so that it cannot be accessed by unauthorized Devices, groups or individuals for either malicious or benign purposes. In addition, since Sensitive Data is often used for authentication and encryption, it must maintain its integrity against intentional or accidental alteration.

A partial list of Sensitive Data is outlined below:

Data	Integrity protection	Confidentiality protection
Owner PSK (Symmetric Keys)	Yes	Yes
Service provisioning keys	Yes	Yes
Asymmetric Private Keys	Yes	Yes
Certificate Data and Signed Hashes	Yes	Not required
Public Keys	Yes	Not required
Access credentials (e.g. SSID, passwords, etc.)	Yes	Yes
ECDH/ECDH Dynamic Shared Key	Yes	Yes
Root CA Public Keys	Yes	Not required
Device and Platform IDs	Yes	Not required

Table 63 – Examples of Sensitive Data

Exact method of protection for secure storage is implementation specific, but typically combinations of hardware and software methods are used.

15.1.1.1 Hardware secure storage

Hardware secure storage is recommended for use with critical Sensitive Data such as symmetric and asymmetric private keys, access credentials, and personal private data. Hardware secure storage most often involves semiconductor-based non-volatile memory ("NVRAM") and includes countermeasures for protecting against unauthorized access to Critical Sensitive Data.

Hardware-based secure storage not only stores Sensitive Data in NVRAM, but also provides protection mechanisms to prevent the retrieval of Sensitive Data through physical and/or electronic attacks. It is not necessary to prevent the attacks themselves, but an attempted attack should not result in an unauthorized entity successfully retrieving Sensitive Data.

Protection mechanisms should provide JIL Moderate protection against access to Sensitive Data from attacks that include but are not limited to:

1) Physical decapping of chip packages to optically read NVRAM contents
2) Physical probing of decapped chip packages to electronically read NVRAM contents
3) Probing of power lines or RF emissions to monitor voltage fluctuations to discern the bit patterns of Critical Sensitive Data
4) Use of malicious software or firmware to read memory contents at rest or in transit within a microcontroller
5) Injection of faults that induce improper Device operation or loss or alteration of Sensitive Data

15.1.1.2 Software Storage

It is generally NOT recommended to rely solely on software and unsecured memory to store Sensitive Data even if it is encrypted. Critical Sensitive Data such as authentication and encryption keys should be housed in hardware secure storage whenever possible.

Sensitive Data stored in volatile and non-volatile memory shall be encrypted using acceptable algorithms to prevent access by unauthorized parties through methods described in Section 15.1.1.1.

15.1.1.3 Additional Security Guidelines and Best Practices

Below are some general practices that can help ensure that Sensitive Data is not compromised by various forms of security attacks:

1) FIPS Random Number Generator ("RNG") – Insufficient randomness or entropy in the RNG used for authentication challenges can substantially degrade security strength. For this reason, it is recommended that a FIPS 800-90A-compliant RNG with a certified noise source be used for all authentication challenges.

2) Secure download and boot – To prevent the loading and execution of malicious software, where it is practical, it is recommended that Secure Download and Secure Boot methods that authenticate a binary's source as well as its contents be used.

3) Deprecated algorithms –Algorithms included but not limited to the list below are considered unsecure and shall not be used for any security-related function:

   a. SHA-1

   b. MD5

   c. RC4

   d. RSA 1024

4) Encrypted transmission between blocks or components – Even if critical Sensitive Data is stored in Secure Storage, any use of that data that requires its transmission out of that Secure Storage should be encrypted to prevent eavesdropping by malicious software within an MCU/MPU.

15.1.2 Secure execution engine

Execution engine is the part of computing Platform that processes security functions, such as cryptographic algorithms or security protocols (e.g. DTLS). Securing the execution engine requires the following

- Isolation of execution of sensitive processes from unauthorized parties/ processes. This includes isolation of CPU caches, and all of execution elements that needed to be considered as part of trusted (crypto) boundary.
- Isolation of data paths into and out of execution engine. For instance both unencrypted but sensitive data prior to encryption or after decryption, or cryptographic keys used for cryptographic algorithms, such as decryption or signing. See trusted paths for more details.

15.1.3 Trusted input/output paths

Paths/ ports used for data entry into or export out of trusted/ crypto-boundary needs to be protected. This includes paths into and out secure execution engine and secure memory.

Path protection can be both hardware based (e.g. use of a privileged bus) or software based (using encryption over an untrusted bus).

15.1.4 Secure clock

Many security functions depend on time-sensitive credentials. Examples are time stamped Kerberos tickets, OAUTH tokens, X.509 certificates, OSCP response, software upgrades, etc. Lack of secure source of clock can mean an attacker can modify the system clock and fool the validation mechanism. Thus an SEE needs to provide a secure source of time that is protected from tampering. Note that trustworthiness from security robustness standpoint is not the same as accuracy. Protocols such as NTP can provide rather accurate time sources from the network, but are not immune to attacks. A secure time source on the other hand can be off by seconds or minutes depending on the time-sensitivity of the corresponding security mechanism. Note that secure time source can be external as long as it is signed by a trusted source and the signature validation in the local Device is a trusted process (e.g. backed by secure boot).

15.1.5 Approved algorithms

An important aspect of security of the entire ecosystem is the robustness of publicly vetted and peer-reviewed (e.g. NIST-approved) cryptographic algorithms. Security is not achieved by obscurity of the cryptographic algorithm. To ensure both interoperability and security, not only widely accepted cryptographic algorithms must be used, but also a list of approved cryptographic functions must be specified explicitly. As new algorithms are NIST approved or old algorithms are deprecated, the list of approved algorithms must be maintained by OCF. All other algorithms (even if they deemed stronger by some parties) must be considered non-approved.

The set of algorithms to be considered for approval are algorithms for

- Hash functions
- Signature algorithms
- Encryption algorithms
- Key exchange algorithms
- Pseudo Random functions (PRF) used for key derivation

This list will be included in this or a separate security robustness rules specification and must be followed for all security specifications within OCF.

15.1.6 Hardware tamper protection

Various levels of hardware tamper protection exist. We borrow FIPS 140-2 terminology (not requirements) regarding tamper protection for cryptographic module

- Production-grade (lowest level): this means components that include conformal sealing coating applied over the module's circuitry to protect against environmental or other physical damage. This does not however require zeroization of secret material during physical maintenance. This definition is borrowed from FIPS 140-2 security level 1.
- Tamper evident/proof (mid-level), This means the Device shows evidence (through covers, enclosures, or seals) of an attempted physical tampering. This definition is borrowed from FIPS 140-2 security level 2.
- Tamper resistance (highest level), this means there is a response to physical tempering that typically includes zerioization of sensitive material on the module. This definition is borrowed from FIPS 140-2 security level 3.

It is difficult of specify quantitative certification test cases for accreditation of these levels. Content protection regimes usually talk about different tools (widely available, specialized and professional tools) used to circumvent the hardware protections put in place by manufacturing. If needed, OCF can follow that model, if and when OCF engage in distributing sensitive key material (e.g. PKI) to its members.

15.2 Secure Boot

15.2.1 Concept of software module authentication

In order to ensure that all components of a Device are operating properly and have not been tampered with, it is best to ensure that the Device is booted properly. There may be multiple stages of boot. The end result is an application running on top an operating system that takes advantage of memory, CPU and peripherals through drivers.

The general concept is the each software module is invoked only after cryptographic integrity verification is complete. The integrity verification relies on the software module having been hashed (e.g. SHA_1, SHA_256) and then signed with a cryptographic signature algorithm with (e.g. RSA), with a key that only a signing authority has access to.

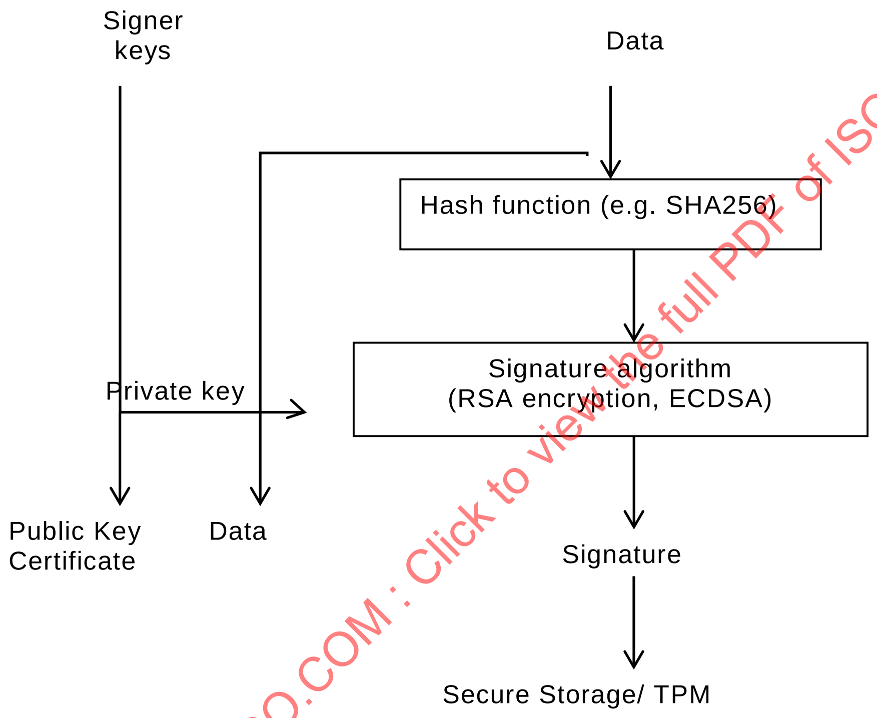


Figure 40 – Software Module Authentication

After the data is signed with the signer's signing key (a private key), the verification key (the public key corresponding to the private signing key) is provided for later verification. For lower level software modules, such as bootloaders, the signatures and verification keys are inserted inside tamper proof memory, such as One time programmable memory or TPM. For higher level software modules, such as application software, the signing is typically performed according to the PKCS#7 format (IETF CMS RFC), where the signedData format includes both indications for signature algorithm, hash algorithm as well as the signature verification key (or certificate). The secure boot specification however does not require use of PKCS#7 format.

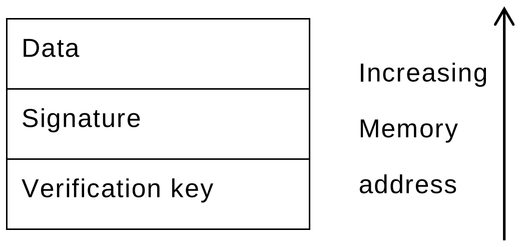


Figure 41 – Verification Software Module

The verification module first decrypts the signature with the verification key (public key of the signer). The verification module also calculates a hash of the data and then compares the decrypted signature (the original) with the hash of data (actual) and if the two values match, the software module is authentic.

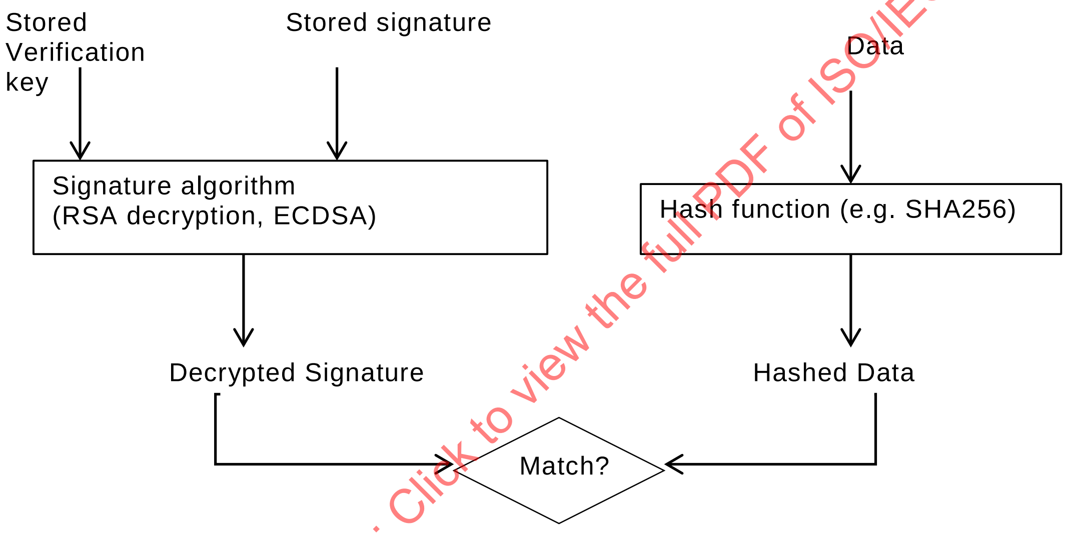


Figure 42 – Software Module Authenticity

15.2.2 Secure Boot process

Depending on the Device implementation, there may be several boot stages. Typically, in a PC/ Linux type environment, the first step is to find and run the BIOS code (first-stage bootloader) to find out where the boot code is and then run the boot code (second-stage boot loader). The second stage bootloader is typically the process that loads the operating system (Kernel) and transfers the execution to the where the Kernel code is. Once the Kernel starts, it may load external Kernel modules and drivers.

When performing a secure boot, it is required that the integrity of each boot loader is verified before executing the boot loader stage. As mentioned, while the signature and verification key for the lowest level bootloader is typically stored in tamper-proof memory, the signature and verification key for higher levels should be embedded (but attached in an easily accessible manner) in the data structures software.

15.2.3 Robustness requirements

To qualify as high robustness secure boot process, the signature and hash algorithms shall be one of the approved algorithms, the signature values and the keys used for verification shall be stored in secure storage and the algorithms shall run inside a secure execution environment and the keys shall be provided the SEE over trusted path.

15.2.3.1 Next steps

Develop a list of approved algorithms and data formats

15.3 Attestation

15.4 Software Update

15.4.1 Overview:

The Device lifecycle does not end at the point when a Device is shipped from the manufacturer; the distribution, retailing, purchase, installation/onboarding, regular operation, maintenance and end-of-life stages for the Device remain outstanding. It is possible for the Device to require update during any of these stages, although the most likely times are during onboarding, regular operation and maintenance. The aspects of the software include, but are not limited to, firmware, operating system, networking stack, application code, drivers, etc.

15.4.2 Recognition of Current Differences

Different manufacturers approach software update utilizing a collection of tools and strategies: over-the-air or wired USB connections, full or partial replacement of existing software, signed and verified code, attestation of the delivery package, verification of the source of the code, package structures for the software, etc.

It is recommended that manufacturers review their processes and technologies for compliance with industry best-practices that a thorough security review of these takes place and that periodic review continue after the initial architecture has been established.

This specification applies to software updates as recommended to be implemented by Devices; it does not have any bearing on the above-mentioned alternative proprietary software update mechanisms.

15.4.3 Software Version Validation

Setting the Initiate Software Version Validation bit in the /oic/sec/pstat.tm Property (see Table 51 of Section 13.7) indicates a request to initiate the software version validation process, the process whereby the Device validates the software (including firmware, operating system, Device drivers, networking stack, etc.) against a trusted source to see if, at the conclusion of the check, the software update process will need to be triggered (see below). When the Initiate Software Version Validation bit of /oic/sec/pstat.tm is set to 1 (TRUE) by a sufficiently privileged Client, the Device sets the /oic/sec/pstat.cm Initiate Software Version Validation bit to 0 and initiates a software version check. Once the Device has determined if an update is available, it sets the Initiate Software Version Validation bit in the /oic/sec/pstat.cm property to 1 (TRUE) if an update is available or 0 (FALSE) if no update is available. To signal completion of the Software Version Validation process, the Device sets the Initiate Software Version Validation bit in the /oic/sec/pstat.tm Property back to 0 (FALSE). If the Initiate Software Version Validation bit of /oic/sec/pstat.tm is set to 0 (FALSE) by a Client, it has no effect on the validation process.

15.4.4 Software Update

Setting the Initiate Secure Software Update bit in the /oic/sec/pstat.tm property (see Table 51 of Section 13.7) indicates a request to initiate the software update process. When the Initiate Secure Software Update bit of /oic/sec/pstat.tm is set to 1 (TRUE) by a sufficiently privileged Client, the Device sets the /oic/sec/pstat.cm Initiate Software Version Validation bit to 0 and initiates a software update process. Once the Device has completed the software update process, it sets the Initiate Secure Software Update bit in the /oic/sec/pstat.cm property to 1 (TRUE) if/when the software was successfully updated or 0 (FALSE) if no update was performed. To signal completion of the Secure Software Update process, the Device sets the Initiate Secure Software Update bit in the /oic/sec/pstat.tm Property back to 0 (FALSE). If the Initiate Secure Software Update bit of /oic/sec/pstat.tm is set to 0 (FALSE) by a Client, it has no effect on the update process.

15.4.5 Recommended Usage

The Initiate Secure Software Update bit of /oic/sec/pstat.tm should only be set by a Client after the Initiate Software Version Validation check is complete.

The process of updating Device software may involve state changes that affect the Device Operational State (/oic/sec/pstat.dos). Devices with an interest in the Device(s) being updated should monitor /oic/sec/pstat.dos and be prepared for pending software update(s) to affect Device state(s) prior to completion of the update.

Note that the Device itself may indicate that it is autonomously initiating a software version check/update or that a check/update is complete by setting the pstat.tm and pstat.cm Initiate Software Version Validation and Secure Software Update bits when starting or completing the version check or update process. As is the case with a Client-initiated update, Clients can be notified that an autonomous version check or software update is pending and/or complete by observing pstat resource changes.

15.5 Non-OCF Endpoint interoperability

15.7 Security Levels

Security Levels are a way to differentiate Devices based on their security criteria. This need for differentiation is based on the requirements from different verticals such as industrial and health care and may extend into smart home. This differentiation is distinct from Device classification (e.g. RFC7228)

These categories of security differentiation may include, but is not limited to:

1. Security Hardening
2. Identity Attestation
3. Certificate/Trust
4. Onboarding Technique
5. Regulatory Compliance
    a. Data at rest
    b. Data in transit
6. Cipher Suites – Crypto Algorithms & Curves
7. Key Length
8. Secure Boot/Update

In the future security levels can be used to define interoperability.

The following applies to Security Specification 1.1:
The current specification does not define any other level beyond Security Level 0. All Devices will be designated as Level 0. Future versions may define additional levels.

Note the following points:

- The definition of a given security level will remain unchanged between versions of the specification.
- Devices that meet a given level may, or may not, be capable of upgrading to a higher level.
- Devices may be evaluated and re-classified at a higher level if it meets the requirements of the higher level (e.g. if a Device is manufactured under the 1.1 version of the specification, and a later spec version defines a security level 1, the Device could be evaluated and classified as level 1 if it meets level 1 requirements).
- The security levels may need to be visible to the end user.

16 Appendix A: Access Control Examples

16.1 Example OCF ACL Resource

The Server is required to verify that any hosted Resource has authorized access by the Client requesting access. The /oic/sec/acl2 Resource is co-located on the Resource host so that the Resource request processing should be applied securely and efficiently. This example shows how a /oic/sec/acl2 Resource could be configured to enforce an example access policy on the Server.

```
{
    "aclist2": [
      {
        // Subject with ID …01 should access two named Resources with access
mode "CRUDN" (Create, Retrieve, Update, Delete and Notify)
        "subject": {"uuid": "XXXX-…-XX01"},
        "resources": [
                      {"href":"/oic/sh/light/1"},
                      {"href":"/oic/sh/temp/0"}
        ],
        "permission": 31, // 31 dec = 0b0001 1111 which maps to ---N DURC
        "validity": [
           // The period starting at 18:00:00 UTC, on January 1, 2015 and
           // ending at 07:00:00 UTC on January 2, 2015
           "period": ["20150101T180000Z/20150102T070000Z"],
           // Repeats the {period} every week until the last day of Jan.
2015.
           "recurrence": ["RRULE:FREQ=WEEKLY;UNTIL=20150131T070000Z"]
          },
        "aceid": 1
      }
    ],
     // An ACL provisioning and management service should be identified as
     // the resource owner
    "rowneruuid": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1"
}
```

16.2 Example Access Manager Service

The AMS should be used to centralize management of access policy, but requires Servers to open a connection to the AMS whenever the named Resources are accessed. This example demonstrates how the /oic/sec/amacl Resource should be configured to achieve this objective.

```
{
  "resources": [
    // If the {Subject} wants to access the /oic/sh/light/1 Resource at host1 and an Amacl was
    // supplied then use the ~~{1}~~ sacl validation credential to enforce access.
    {"href": /oic/sh/light/1},
    // If the {Subject} wants to access the /oma/3 Resource at host2 and an AM sacl was
    // supplied  then use the ~~{1}~~ sacl validation credential to enforce access.
    {"href": "/oma/3"},
    // If the {Subject} wants to access any local Resource and an Amacl was supplied then use
    // the ~~{1}~~ sacl validation credential to enforce access.
    {"wc": "*"}]
}
```

17 Appendix B: Execution Environment Security Profiles

Given that IoT verticals and Devices will not be of uniform capabilities, a one-size-fits all security robustness requirements meeting all IOT applications and services will not serve the needs of OCF, and security profiles of varying degree of robustness (trustworthiness), cost and complexity have to be defined. To address a large ecosystem of vendors, the profiles can only be defined as requirements and the exact solutions meeting those requirements are specific to the vendors' open or proprietary implementations, and thus in most part outside scope of this document.

To align with the rest of OCF specifications, where Device classifications follow IETF RFC 7228 (Terminology for constrained node networks) methodology, we limit the number of security profiles to a maximum of 3. However, our understanding is OCF capabilities criteria for each of 3 classes will be more fit to the current IoT chip market than that of IETF.

Given the extremely low level of resources at class 0, our expectation is that class 0 Devices are either capable of no security functionality or easily breakable security that depend on environmental (e.g. availability of human) factors to perform security functions. This means the class 0 will not be equipped with an SEE.

Platform class	SEE	Robustness level
0	No	N/A
1	Yes	Low
2	Yes	High

Table 64 – OCF Security Profile

Technical Note: This analysis acknowledges that these Platform classifications do not take into consideration of possibility of security co-processor or other hardware security capability that augments classification criteria (namely CPU speed, memory, storage).

18 Appendix C: RAML Definition

All the sub-clauses in Appendix C describe the Security Resources with a restful API definition language. The Resources presented in Appendix C are formatted for readability, and so may appear to have extra line breaks. The contents of the Resource Types without the extra line breaks are available in OCF Resource Type Definitions.

Resource Name	Resource Type	Section
Access Control List	oic.r.acl	A.1
Access Control List 2	oic.r.acl2	A.2
Managed Access Control List	oic.r.amacl	A.3
Signed Access Control List	oic.r.sacl	A.4
Device Ownership Transfer	oic.r.doxm	A.5
Device Provisioning Status	oic.r.pstat	A.6
Credential	oic.r.cred	A.7
Certificate Signing Request	oic.r.csr	A.8
Roles	oic.r.roles	A.9
Certificate Revocation List	oic.r.crl	A.10

Table 65 – OCF SVR RAML

A.1 OICSecurityAclResource

A.1.1 Introduction

This resource specifies the local access control list.

A.1.2 Example URI

/oic/sec/acl

A.1.3 Resource Type

A.1.4 RAML Definition

```
#%RAML 0.8

title: OICSecurityAclResource
version: v1.1-20161213

traits:
 - interface :
     queryParameters:

       if:
         enum: ["oic.if.baseline"]

 - ace-filtered :
      queryParameters:
        subjectuuid:


/oic/sec/acl:
  description: |
    This resource specifies the local access control list.

  is : ['interface']
  get:
    description: |
      Retrieves the ACL entries.
      When used without query parameters, all the ACE entries are returned.
      When used with a subjectuuid, only the ACEs with the specified
      subjectuuid are returned
      If subjectuuid and resources are specified,
      only the ACEs with the specified subjectuuid and resource hrefs are
      returned.

    is : ['ace-filtered']
    responses :
      200:
        body:
          application/json:
            schema: |
              {
                "$schema": "http://json-schema.org/draft-04/schema#",
                "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.r.acl.json#",
                "title": "Access Control List information",
                "definitions": {
                  "oic.r.acl": {
                    "type": "object",
                    "properties": {
                      "aclist": {
                        "type": "object",
                        "description": "Subject-based Access Control Entries in the ACL resource",
                        "properties": {
                          "aces": {
                            "type": "array",
                            "items": {
                              "$ref": "oic.sec.ace.json#/definitions/oic.sec.ace"
                            }
                          }
                        },
                        "required": [ "aces" ]
                      },
                      "rowneruuid": {
                        "description": "The value identifies the unique resource owner",
                        "$ref": "../../core/schemas/oic.types-schema.json#/definitions/uuid"
                      }
                    }
                  }
                },
                "type": "object",
                "allOf": [
                  { "$ref": "../../core/schemas/oic.core-schema.json#/definitions/oic.core" },
                  { "$ref": "#/definitions/oic.r.acl" }
                ],
                "required": [ "aclist", "rowneruuid" ]
              }

            example: |
              {
                "aclist": {
                  "aces": [
                    {
                      "subjectuuid": "e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
                      "resources": [
                        {
                          "href": "coaps://IP-ADDR/temp",
                          "rel": "some-rel",
                          "rt": ["oic.r.temperature"],
                          "if": ["oic.if.a"]
                        },
                        {
                          "href": "coaps://IP-ADDR/temp",
                          "rel": "some-rel",
                          "rt": ["oic.r.temperature"],
                          "if": ["oic.if.s"]
                        }
                      ],
                      "permission": 31,
                      "validity": [
                        {
                          "period":"20160101T180000Z/20170102T070000Z",
                          "recurrence": [ "DSTART:XXXXX",
"RRULE:FREQ=DAILY;UNTIL=20180131T140000Z;BYMONTH=1" ]
                        },
                        {
                          "period":"20160101T180000Z/PT5H30M",
                          "recurrence": [ "RRULE:FREQ=DAILY;UNTIL=20180131T140000Z;BYMONTH=1" ]
                        }
                      ]
                    }
                  ]
                },
                "rowneruuid": "de305d54-75b4-431b-adb2-eb6b9e546014"
              }

      400:

        description: |
          The request is invalid.

  post:

    description: |
      Updates the ACL resource with the provided values
      ACEs provided
      in the update not currently in the ACL are added
      ACEs that already
      exist in the ACL are ignored.
      Note that for the purposes of update, equivalency is determined
      by comparing the ACE subjectuuid, permission, string comparisons
      of all validity elements, and string comparisons of all resource
      hrefs.

    body:
      application/json:
        schema: |
          {
            "$schema": "http://json-schema.org/draft-04/schema#",
            "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.r.acl.json#",
            "title": "Access Control List information",
            "definitions": {
              "oic.r.acl": {
                "type": "object",
                "properties": {
                  "aclist": {
                    "type": "object",
                    "description": "Subject-based Access Control Entries in the ACL resource",
                    "properties": {
                      "aces": {
                        "type": "array",
                        "items": {
                          "$ref": "oic.sec.ace.json#/definitions/oic.sec.ace"
                        }
                      }
                    },
                    "required": [ "aces" ]
                  },
                  "rowneruuid": {
                    "description": "The value identifies the unique resource owner",
                    "$ref": "../../core/schemas/oic.types-schema.json#/definitions/uuid"
                  }
                }
              }
            },
            "type": "object",
            "allOf": [
              { "$ref": "../../core/schemas/oic.core-schema.json#/definitions/oic.core" },
              { "$ref": "#/definitions/oic.r.acl" }
            ],
            "required": [ "aclist", "rowneruuid" ]
          }

        example: |
          {
            "aclist": {
              "aces": [
                {
                  "subjectuuid": "e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
                  "resources": [
                    {
                      "href": "coaps://IP-ADDR/temp",
                      "rel": "some-rel",
                      "rt": ["oic.r.temperature"],
                      "if": ["oic.if.a"]
                    },
                    {
                      "href": "coaps://IP-ADDR/temp",
                      "rel": "some-rel",
                      "rt": ["oic.r.temperature"],
                      "if": ["oic.if.s"]
                    }
                  ],
                  "permission": 31,
                  "validity": [
                    {
                      "period":"20160101T180000Z/20170102T070000Z",
                      "recurrence": [ "DSTART:XXXXX",
"RRULE:FREQ=DAILY;UNTIL=20180131T140000Z;BYMONTH=1" ]
                    },
                    {
                      "period":"20160101T180000Z/PT5H30M",
                      "recurrence": [ "RRULE:FREQ=DAILY;UNTIL=20180131T140000Z;BYMONTH=1" ]
                    }
                  ]
                }
              ]
            },
            "rowneruuid": "de305d54-75b4-431b-adb2-eb6b9e546014"
          }

    responses :
      400:
        description: |
          The request is invalid.
      201:

        description: |
          The ACL entry/entries is/are created.


      204:

        description: |
          The ACL entry/entries is/are updated.


  delete:

    description: |
      Deletes ACL entries.
      When DELETE is used without query parameters, all the ACE entries are deleted.
      When DELETE is used with a subjectuuid, only the ACEs with the specified
      subjectuuid are deleted
      If subjectuuid and resources are specified,
      only the ACEs with the specified subjectuuid and resource hrefs are
      deleted.

    is : ['ace-filtered']

    responses :

      200:

        description: |
          The matching ACEs or the entire ACL resource has been successfully deleted.


      400:

        description: |
          The request is invalid.
```

A.1.5 Property Definition

Property name	Value type	Mandatory	Access mode	Description
rowneruuid	multiple types: see schema	yes		The value identifies the unique resource owner
aclist	object: see schema	yes		Subject-based Access Control Entries in the ACL resource
aces (aclist)	array: see schema	yes		

A.1.6 CRUDN behavior

Resource	Create	Read	Update	Delete	Notify
/oic/sec/acl		get	post	delete	

A.2 OICSecurityAcl2Resource

A.2.1 Introduction

This resource specifies the local access control list.

A.2.2 Example URI

/oic/sec/acl2

A.2.3 Resource Type

A.2.4 RAML Definition

```
#%RAML 0.8
title: OICSecurityAcl2Resource
version: v1.0-20161214
traits:
 - interface :
     queryParameters:
       if:
         enum: ["oic.if.baseline"]
 - ace-filtered :
     queryParameters:
       aceid:


/oic/sec/acl2:
  description: |
    This resource specifies the local access control list.

  is : ['interface']
  get:
    description: |
      Retrieves the ACL data.
      When used without query parameters, all the ACE entries are returned.
      When used with a query parameter, only the ACEs matching the specified
      parameter are returned.

    is : ['ace-filtered']
    responses :
      200:
        body:
          application/json:
            schema: |
              {
                "$schema": "http://json-schema.org/draft-04/schema#",
                "id": "https://www.openconnectivity.org/ocf-
apis/security/schemas/oic.r.acl2.json#",
                "title": "Access Control List information",
                "definitions": {
                  "oic.r.acl2": {
                    "type": "object",
                    "properties": {
                      "aclist2": {
                        "type": "array",
                        "description": "Access Control Entries in the ACL resource",
                        "items": {
                          "$ref": "oic.sec.ace2.json#/definitions/oic.sec.ace2"
                        }
                      },
                      "rowneruuid": {
                        "description": "The value identifies the unique resource owner",
                        "$ref": "../../core/schemas/oic.types-schema.json#/definitions/uuid"
                      }
                    }
                  }
                },
                "type": "object",
                "allOf": [
                  { "$ref": "../../core/schemas/oic.core-schema.json#/definitions/oic.core" },
                  { "$ref": "#/definitions/oic.r.acl2" }
                ],
                "required": [ "aclist2", "rowneruuid"]
              }

            example: |
              {
                "aclist2": [
                  {
                    "aceid": 1,
                    "subject": {
                      "authority": "484b8a51-cb23-46c0-a5f1-b4aebef50ebe",
                      "role": "SOME_STRING"
                    },
                    "resources": [
                      {
                        "href": "/light",
                        "rt": ["oic.r.light"],
                        "if": ["oic.if.baseline", "oic.if.a"]
                      },
                      {
                        "href": "/door",
                        "rt": ["oic.r.door"],
                        "if": ["oic.if.baseline", "oic.if.a"]
                      }
                    ],
                    "permission": 24
                  },
                  {
                    "aceid": 2,
                    "subject": {
                      "uuid": "e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9"
                    },
                    "resources": [
                      {
                        "href": "/light",
                        "rt": ["oic.r.light"],
                        "if": ["oic.if.baseline", "oic.if.a"]
                      },
                      {
                        "href": "/door",
                        "rt": ["oic.r.door"],
                        "if": ["oic.if.baseline", "oic.if.a"]
                      }
                    ],
                    "permission": 24
                    },
                    {
                      "aceid": 3,
                      "subject": {"conntype": "anon-clear"},
                      "resources": [
                        {
                          "href": "/light",
                          "rt": ["oic.r.light"],
                          "if": ["oic.if.baseline", "oic.if.a"]
                        },
                        {
                          "href": "/door",
                          "rt": ["oic.r.door"],
                          "if": ["oic.if.baseline", "oic.if.a"]
                        }
                      ],
                      "permission": 16,
                      "validity": [
                        {
                          "period":"20160101T180000Z/20170102T070000Z",
                          "recurrence": [ "DSTART:XXXXX",
"RRULE:FREQ=DAILY;UNTIL=20180131T140000Z;BYMONTH=1" ]
                        },
                        {
                          "period":"20160101T180000Z/PT5H30M",
                          "recurrence": [ "RRULE:FREQ=DAILY;UNTIL=20180131T140000Z;BYMONTH=1" ]
                }
              ]
            }
          ],
          "rowneruuid": "de305d54-75b4-431b-adb2-eb6b9e546014"
        }

    400:

      description: |
        The request is invalid.

  post:

    description: |
      Updates the ACL resource with the provided ACEs.
      ACEs provided in the update with aceids not currently in the ACL
      resource are added.
      ACEs provided in the update with aceid(s) already in the ACL completely
      replace the ACE(s) in the ACL resource.
      ACEs provided in the update without aceid properties are added and
      assigned unique aceids in the ACL resource.

    body:
      application/json:

        schema: |

          {
            "$schema": "http://json-schema.org/draft-04/schema#",
            "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.r.acl2.json#",
            "title": "Access Control List information",
            "definitions": {
              "oic.r.acl2": {
                "type": "object",
                "properties": {
                  "aclist2": {
                    "type": "array",
                    "description": "Access Control Entries in the ACL resource",
                    "items": {
                      "$ref": "oic.sec.ace2.json#/definitions/oic.sec.ace2"
                    }
                  },
                  "rowneruuid": {
                    "description": "The value identifies the unique resource owner",
                    "$ref": "../../core/schemas/oic.types-schema.json#/definitions/uuid"
                  }
                }
              }
            },
            "type": "object",
            "allOf": [
              { "$ref": "../../core/schemas/oic.core-schema.json#/definitions/oic.core" },
              { "$ref": "#/definitions/oic.r.acl2" }
            ],
            "required": [ "aclist2", "rowneruuid"]
          }

        example: |

          {
            "aclist2": [
              {
                "aceid": 1,
                "subject": {
                  "authority": "484b8a51-cb23-46c0-a5f1-b4aebef50ebe",
                  "role": "SOME_STRING"
                },
                "resources": [
                  {
                    "href": "/light",
                    "rt": ["oic.r.light"],
                    "if": ["oic.if.baseline", "oic.if.a"]
                  },
                  {
                    "href": "/door",
                    "rt": ["oic.r.door"],
                    "if": ["oic.if.baseline", "oic.if.a"]
                  }
                ],
                "permission": 24
              },
              {
                "aceid": 3,
                "subject": {
                  "uuid": "e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9"
                },
                "resources": [
                  {
                    "href": "/light",
                    "rt": ["oic.r.light"],
                    "if": ["oic.if.baseline", "oic.if.a"]
                  },
                  {
                    "href": "/door",
                    "rt": ["oic.r.door"],
                    "if": ["oic.if.baseline", "oic.if.a"]
                  }
                ],
                "permission": 24
              }
            ],
            "rowneruuid": "e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9"
          }

    responses :

      400:

        description: |
          The request is invalid.


      201:

        description: |
          The ACL entry is created.


      204:

        description: |
          The ACL entry is updated.


  delete:

    description: |
      Deletes ACL entries.
      When DELETE is used without query parameters, all the ACE entries are deleted.
      When DELETE is used with a query parameter, only the ACEs matching the
      specified parameter are deleted.


    is : ['ace-filtered']

    responses :

      200:

        description: |
          The matching ACEs or the entire ACL resource has been successfully deleted.


      400:

        description: |
  The request is invalid.
```

A.2.5 Property Definition

Property name	Value type	Mandatory	Access mode	Description
rowneruuid	multiple types: see schema	yes		The value identifies the unique resource owner
aclist2	array: see schema	yes		Access Control Entries in the ACL resource

A.2.6 CRUDN behavior

Resource	Create	Read	Update	Delete	Notify
/oic/sec/acl2		get	post	delete	

A.2.7 Referenced JSON schemas

A.2.8 oic.sec.didtype.json

A.2.9 Property Definition

Property name	Value type	Mandatory	Access mode	Description
uuid	multiple types: see schema	yes		A UUID Device ID

A.2.10 Schema Definition

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.sec.didtype.json#",
  "title": "Device Identifier Format type",
  "definitions": {
    "oic.sec.didtype": {
          "type": "object",
          "description": "Device identifier",
          "properties": {
            "uuid": {
              "description": "A UUID Device ID",
              "$ref": "../../core/schemas/oic.types-schema.json#/definitions/uuid"
            }
          },
          "required": ["uuid"]
        }
    }
}
```

A.2.11 oic.sec.ace2.json

A.2.12 Property Definition

Property name	Value type	Mandatory	Access mode	Description
aceid	integer			An identifier for the ACE that is unique within the ACL. In cases where it isn't supplied in an update, the Server will add the ACE and

				assign it a unique value.
subject	multiple types: see schema	yes		The subject to whom this ace applies, either a deviceId, role, or wildcard
validity	array: see schema			validity is an array of time-pattern objects
resources	array: see schema			References the application's resources to which a security policy applies
rt (resources)	multiple types: see schema			When present, the ACE only applies when the rt (resource type) matches
href (resources)	multiple types: see schema			When present, the ACE only applies when the href matches
wc (resources)	string			A wildcard matching policy
if (resources)	multiple types: see schema			When present, the ACE only applies when the if (interface) matches
permission	integer	yes		Bitmask encoding of CRUDN permission

A.2.13 Schema Definition

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.sec.ace2.json#",
  "title": "Subject-based Access Control Entry (ACE) object definition",
  "definitions": {
    "oic.sec.ace2": {
      "type": "object",
      "properties": {
        "aceid": {
          "type": "integer",
          "minimum": 1,
          "description": "An identifier for the ACE that is unique within the ACL. In cases where
it isn't supplied in an update, the Server will add the ACE and assign it a unique value."
        },
        "resources": {
          "type": "array",
          "description": "References the application's resources to which a security policy
applies",
          "items": {
            "type": "object",
            "description": "Each resource must have at least one of these properties set",
            "properties": {
              "href": {
                "description": "When present, the ACE only applies when the href matches",
                "$ref": "oic.oic-link-schema.json#/definitions/oic.oic-link/properties/href"
              },
              "rt": {
                "description": "When present, the ACE only applies when the rt (resource type)
matches",
                "$ref": "oic.oic-link-schema.json#/definitions/oic.oic-link/properties/rt"
              },
              "if": {
                "description": "When present, the ACE only applies when the if (interface)
matches",
                "$ref": "oic.oic-link-schema.json#/definitions/oic.oic-link/properties/if"
              },
              "wc": {
                "type": "string",
                "enum": [ "+", "-", "*" ],
                "description": "A wildcard matching policy",
                "detail-desc": [  "+ - Matches all discoverable resources",
                                  "- - Matches all non-discoverable resources",
                                  "* - Matches all resources"
                ]
              }
            }
          }
        },
        "permission": {
          "type": "integer",
          "description": "Bitmask encoding of CRUDN permission",
          "$ref": "oic.sec.crudntype.json#/definitions/oic.sec.crudntype/properties/bitmask"
        },
        "subject": {
          "description": "The subject to whom this ace applies, either a deviceId, role, or
wildcard",
          "anyOf": [
            {
              "$ref": "oic.sec.didtype.json#/definitions/oic.sec.didtype"
            },
            {
              "$ref": "oic.sec.roletype.json#/definitions/oic.sec.roletype"
            },
            {
              "type": "object",
              "properties": {
                "conntype": {
                  "type": "string",
                  "enum": [ "auth-crypt", "anon-clear" ],
                  "description": "This property allows an ACE to be matched based on the connection
or message type",
                  "detail-desc": [  "auth-crypt - ACE applies if the Client is authenticated and
the data channel or message is encrypted and integrity protected",
                                    "anon-clear - ACE applies if the Client is not authenticated
and the data channel or message is not encrypted but may be integrity protected"
                  ]
                }
              },
              "required": ["conntype"]
            }
          ]
        },
        "validity": {
          "type": "array",
          "description": "validity is an array of time-pattern objects",
          "items": {
            "$ref": "oic.sec.time-pattern.json#/definitions/time-pattern"
          }
        }
      },
      "required": [ "permission", "subject" ]
    }
  }
}
```

A.2.14 oic.sec.roletype.json

A.2.15 Property Definition

Property name	Value type	Mandatory	Access mode	Description
role	string	yes		The ID of the role being identified.
authority	string			The Authority component of the entity being identified. A NULL <Authority> refers to the local entity or device.

A.2.16 Schema Definition

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.sec.roletype.json#",
  "title": "Security Role Types",
  "definitions": {
    "oic.sec.roletype": {
      "type": "object",
      "description": "Security role specified as an <Authority> & <Rolename>. A NULL <Authority>
refers to the local entity or device.",
      "properties": {
        "authority": {
          "type": "string",
          "description": "The Authority component of the entity being identified. A NULL
<Authority> refers to the local entity or device."
        },
        "role": {
          "type": "string",
          "description": "The ID of the role being identified."
        }
      },
      "required": ["role"]
    }
  }
}
```

A.2.17 oic.sec.time-pattern.json

A.2.18 Property Definition

Property name	Value type	Mandatory	Access mode	Description
recurrence	array: see schema			String array represents a recurrence rule using the RFC5545 Recurrence
period	string	yes		String represents a period using the RFC5545 Period

A.2.19 Schema Definition

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.sec.time-pattern.json#",
  "title": "RFC5545 time period and recurrence rule(s)",
  "definitions": {
    "time-pattern": {
      "type": "object",
      "description": "The time-pattern contains a period and recurrence expressed in RFC5545
syntax",
      "properties": {
        "period": {
          "type": "string",
          "description": "String represents a period using the RFC5545 Period"
        },
        "recurrence": {
          "type": "array",
          "description": "String array represents a recurrence rule using the RFC5545 Recurrence",
          "items": {
            "type": "string"
          }
        }
      },
      "required": [ "period" ]
    }
  }
}
```

A.2.20 oic.sec.crudntype.json

A.2.21 Property Definition

Property name	Value type	Mandatory	Access mode	Description
bitmask	integer	yes		The encoded bitmask indicating permissions

A.2.22 Schema Definition

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.sec.crudntype.json#",
  "title": "Permission BitMask",
  "definitions": {
    "oic.sec.crudntype": {
      "description": "OIC CRUDN types",
      "properties": {
        "bitmask": {
          "type": "integer",
          "minimum": 0,
          "maximum": 31,
          "description": "The encoded bitmask indicating permissions",
          "detail-desc": [ "0  - No permissions",
                           "1 - Create permission is granted",
                           "2 - Read, observe, discover permission is granted",
                           "4 - Write, update permission is granted",
                           "8 - Delete permission is granted",
                           "16 - Notify permission is granted" ]
        }
      }
    }
  },
  "type": "object",
  "allOf": [
    { "$ref": "#/definitions/oic.sec.crudntype" }
  ],
  "required": ["bitmask"]
}
```

A.3 OICSecurityAmaclResource

A.3.1 Introduction

This resource specifies the host resources with access permission that is managed by an AMS.

A.3.2 Example URI

/oic/sec/amacl

A.3.3 Resource Type

A.3.4 RAML Definition

```
#%RAML 0.8

title: OICSecurityAmaclResource
version: v1.0-20150819

traits:
 - interface :
     queryParameters:

       if:
         enum: ["oic.if.baseline"]


/oic/sec/amacl:

  description: |
    This resource specifies the host resources with access permission that is managed by an AMS.

  is : ['interface']
  get:

    description: |
      Retrieves the amacl data.

    responses :

      200:

        body:
          application/json:

            schema: |
              {
                "$schema": "http://json-schema.org/draft-04/schema#",
                "id": "https://www.openconnectivity.org/ocf-
apis/security/schemas/oic.r.amacl.json#",
                "title": "Managed Access Control information",
                "definitions": {
                  "oic.r.amacl": {
                    "type": "object",
                    "properties": {
                      "resources": {
                        "type": "array",
                        "description": "Multiple links to this host's resources",
                        "items": { "$ref": "oic.sec.ace2.json#/properties/resources" }
                      }
                    }
                  }
                },
                "type": "object",
                "allOf": [
                  { "$ref": "#/definitions/oic.r.amacl" }
                ],
                "required": [ "resources" ]
              }

            example: |
              {
                "resources": [
                  {
                    "href": "/temp",
                    "rt": ["oic.r.temperature"],
                    "if": ["oic.if.baseline", "oic.if.a"]
                  },
                  {
                    "href": "/temp",
                    "rt": ["oic.r.temperature"],
                    "if": ["oic.if.baseline", "oic.if.s"]
                  }
                ]
              }

  post:

    description: |
      Sets the new amacl data

    body:
      application/json:

        schema: |

          {
            "$schema": "http://json-schema.org/draft-04/schema#",
            "id": "https://www.openconnectivity.org/ocf-apis/security/schemas/oic.r.amacl.json#",
            "title": "Managed Access Control information",
            "definitions": {
              "oic.r.amacl": {
                "type": "object",
                "properties": {
                  "resources": {
                    "type": "array",
                    "description": "Multiple links to this host's resources",
                    "items": { "$ref": "oic.sec.ace2.json#/properties/resources" }
                  }
                }
              }
            },
            "type": "object",
            "allOf": [
              { "$ref": "#/definitions/oic.r.amacl" }
            ],
            "required": [ "resources" ]
          }

        example: |

          {
            "resources": [
              {
                "href": "/temp",
                "rt": ["oic.r.temperature"],
                "if": ["oic.if.baseline", "oic.if.a"]
              },
              {
                "href": "/temp",
                "rt": ["oic.r.temperature"],
                "if": ["oic.if.baseline", "oic.if.s"]
              }
            ]
          }

    responses :

      400:

        description: |
```