OIC SECURITY SPECIFICATION V1.0.0

Open Interconnect Consortium (OIC)
admin@openinterconnect.org

Legal Disclaimer

CONTENTS

1 Scope

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

OIC Core Specification, version 1.0, Open Interconnect Consortium, June 13, 2015. Available at: <link to be added>. Latest version available at: <link to be added>.

OIC Smart Home Resource Specification, version 1.0, Open Interconnect Consortium, June 13, 2015. Available at: <link to be added>. Latest version available at: <link to be added>.

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.

3 Terms, Definitions, Symbols and Abbreviations

Terms, definitions, symbols and abbreviations used in this specification are defined by the OIC 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 OIC specifications as a convenience for the reader. It is considered non-normative.

3.1 Terms and definitions

Term	Description
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 an OIC Server which allows or denies a pending access request.
ACL Provisioning Service	A name and resource type (oic.sec.aps) given to an OIC device that is authorized to provision ACL resources.

Action	A sequence of commands intended for OIC servers
Bootstrap Service	An OIC device that implements a service of type oic.sec.bss
Bootstrap and provisioning tool	A logical entity handling initial provisioning of security (e.g. credentials) into a newly introduced device.
OIC Client	OIC stack instance and application. Typically, the OIC Client performs actions involving resources hosted by OIC Servers.
Credential Management Service	A name and resource type (oic.sec.cms) given to an OIC device that is authorized to provision credential resources.
OIC Device	An instance of an OIC stack. Multiple stack instances may exist on the same platform.
Device Class	RFC 7228 defines classes of constrained devices that distinguishes when the OIC small footprint stack is used vs. a large footprint stack. Class 2 and below is for small footprint stacks.
Entity	An element of the physical world that is exposed through an OIC Device
DeviceID	OIC stack instance identifier.
Interface	Interfaces define expected parameters to GET, PUT, POST, DELETE commands for specific resources
Intermediary	A device that implements both client and server roles and may perform protocol translation, virtual device to physical device mapping or resource translation.
OIC Cipher Suite	A set of algorithms and parameters that define the cryptographic functionality of an OIC Device. The OIC Cipher Suite includes the definition of the public key group operations, signatures, and specific hashing and encoding used to support the public key. An OIC Cipher Suite should include a DTLS cipher suite.
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.
PlatformID	Uniquely identifies the platform consisting of hardware, firmware and operating system. The platform ID is considered unique and immutable and typically inserted in platform in an integrity protected manner. A platform may host multiple OIC Devices.
Property	A named data element within a resource. May refer to intrinsic properties that are common across all OIC resources.
Resource	A data structure that defines the properties, type and interfaces of an OIC Device.
Role (network context)	Stereotyped behavior of an OIC device; one of [Client, Server or Intermediary]
Role (Security context)	A property of an OIC credential resource that names a role that a device may assert when attempting access to device resources. Access policies may differ for OIC Client if access is attempted

	through a role vs. the device UUID. This document assumes the security context unless otherwise stated.
OIC Server	An OIC resource host.
Secure Resource Manager	A module in the OIC Core that implements security functionality that includes management of security resources such as ACLs, credentials and device owner transfer state.
SACL	A signed ACL resource that is dynamically supplied to an OIC Server
Trust Anchor	A well-defined, shared authority, within a trust hierarchy, by which two cryptographic entities (e.g. an OIC device and an onboarding tool) can assume trust.
Unique Authenticable Identifier	A unique identifier created from the hash of a public key and associated OIC Cipher Suite that is used to create the DeviceID. The ownership of a UAID may be authenticated by peer devices.

Table 1 – Terminology

3.2 Symbols and Abbreviations

Symbol	Description
ACL	Access control list
AMS	Access manager service
APS	ACL provisioning service
BPT	Bootstrap and provisioning Tool
BSS	Bootstrap service
CMS	Credential management service
CRUDN	Create, Read, Update, Delete, Notify
OBT	Onboarding Tool
SRM	Secure Resource Manager
UAID	Unique Authenticable IDentifer

Table 2 - Symbols and abbreviations

3.3 Conventions

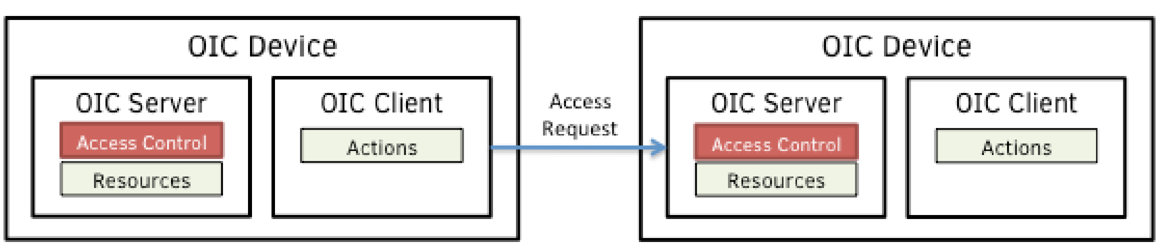


Figure 1 - OIC interactions

OIC devices may implement OIC Client role that performs Actions on OIC Servers. Actions access Resources managed by OIC Servers. The OIC 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 OIC core framework and applications.

For the purposes of this document, the terms and definitions given in OIC 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 OIC 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 OIC Core Architecture and should be implemented. Recommended features take advantage of the capabilities OIC 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 OIC 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 OIC Core Specification.

4.3 Document structure

The Smart Home Device specification defines an OIC Device for usage in the Smart Home vertical. This document describes an OIC Device and makes use of functionality defined in the OIC Core Specification.

The OIC Core Specification provides building blocks to define OIC Devices. The following Core functionality is used:

- Required OIC Core Resources.
- Required transports.

Note that other mandatory functions in the Core might be needed to create an OIC compliant device, but are not mentioned in this document.

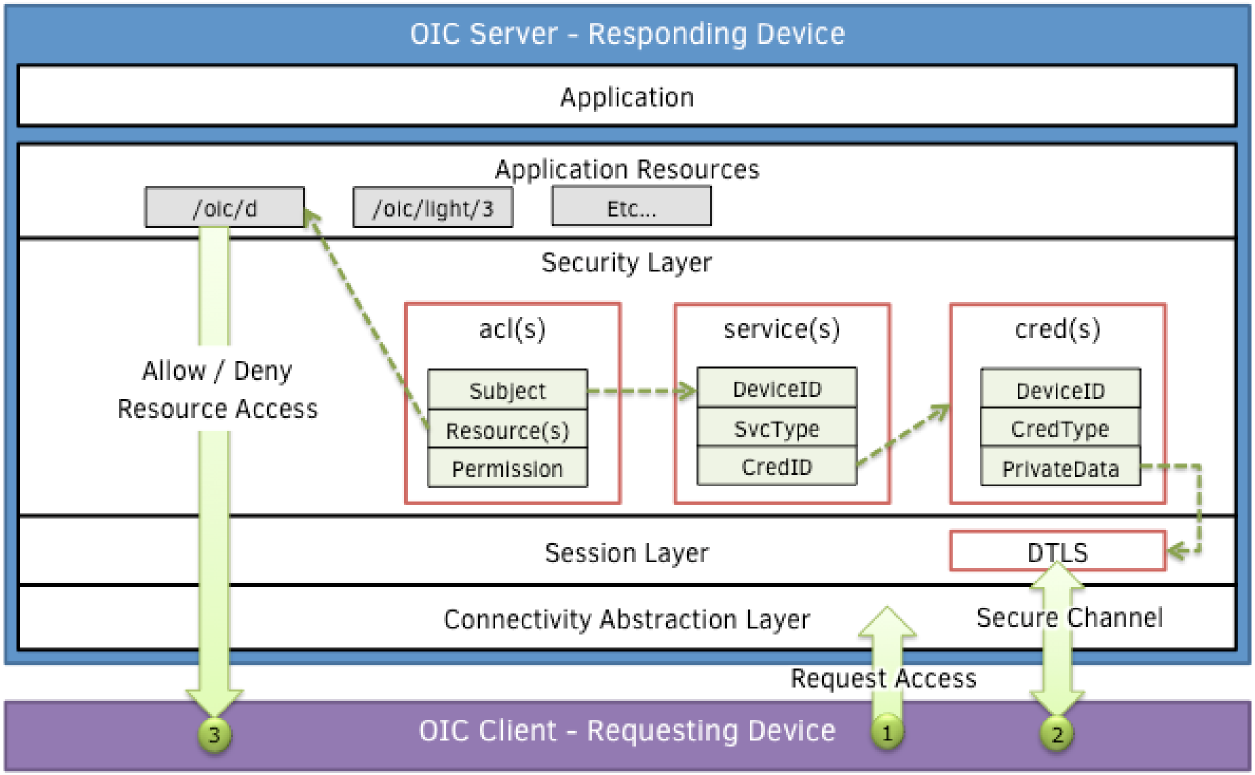
The Security specification may use RAML as a specification language and JSON Schemas as payload definitions for all CRUDN actions. The mapping of the CRUDN actions is specified in the OIC Core Specification.

4.4 Document Sections

5 Security Overview (Informative)

The goal for the OIC security architecture is to protect OIC resources themselves and all aspects of HW and SW that are used to support the protection of OIC resource. From OIC perspective an OIC device is a logical entity that conforms to OIC specifications. The OIC server holds and controls the resources and provides OIC client access to those resources, subject to a set of security mechanisms. The platform, hosting the OIC 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 three steps.



Step-1 - The OIC Client establishes a network connection to the OIC Server (OIC device holding the resources). The connectivity abstraction layer ensures the devices are able to connect despite differences in connectivity options. OIC Devices are identified using a DeviceID, which is different from a platform ID. The platform ID is meant to uniquely identify the physical device. There should be a binding between the device context and the platform implementing the device. Network addresses map to DeviceIDs. The network address is used to establish connectivity, but security policy is expressed in terms of DeviceID.

Note: Future versions of this specification will add a binding between a device (and device ID) and and a platform ID.

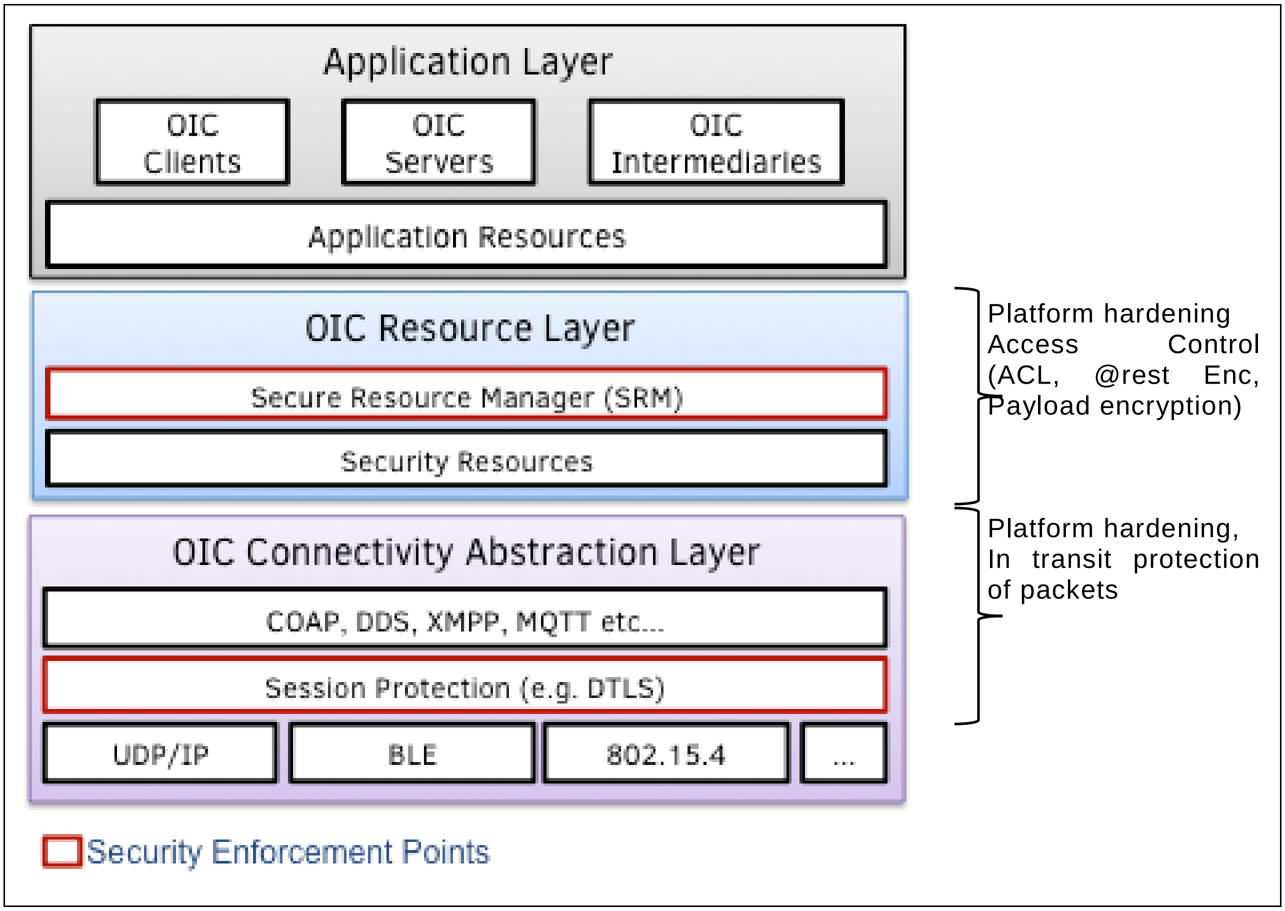
Step-2 - The second step establishes a secure end-to-end channel that protects the exchange of OIC messages and resources passed between OIC devices (e.g. OIC servers and OIC devices). Encryption keys are stored securely (robustness dependent upon platform availability) in the local platform. The OIC *credential* resource is used to reference the encryption keys. The set of devices the OIC Server is able to communicate with securely is contained in the OIC *services* resource. To access any resources on the OIC server, the OIC client must first be authenticated to the OIC server. The OIC server then consults the ACL pertaining to the OIC resource, to which access is being attempted and looks for an ACL entry that matches the OIC client deviceID or roleID. In certain cases, the requester may assert a role, if privileged access is required.

Step 3 – The final step applies the ACL permission to the requested resource where the decision to allow or deny access is enforced by the OIC Server's Secure Resource manager (SRM).

OIC resource protection includes protection of data both while at rest and during transit. It should be noted that, aside from access control mechanisms, OIC security specification does not include specification of secure storage of OIC resources, while stored at OIC 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 14 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. OIC 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.



5.1 Access control (Informative)

OIC framework assumes that resources are hosted at OIC server and are made available to OIC clients subject to access control and authorization mechanisms. The resources at the end point are protected through implementation of access control, authentication (data integrity protection and possibly origin verification) and confidentiality protection. This section provide an overview of access control (AC) through the use of Access Control Lists (ACLs), while leaving other mechanisms such as resource integrity protection, confidentiality protection to other sections. However, AC in the OIC stack is expected to be transport and connectivity-mechanism agnostic

Implementation of access control relies on a-priori definition of a set of access policies for data (object) that needs protection. The policies may be stored by a local ACL or an Access Manager service in form of Access Control Entries (ACE), where each ACE defines permissions required to access a specific object along with the 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 object (data that is to be accessed). 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 object to a role taken by the entity requesting access. Asserting the role of the requestor requires proper authorization process.

In OIC access control model, each resource instance is required to have an associated access control policy. This means, each OIC device acting as OIC server, needs to have an ACL for each resource it is protecting. If access control is SBAC, then there needs to be an ACE for each subject (identity of an OIC client) that needs to access a SBAC controlled resource. However, ACLs for unknown or anonymous (unauthenticated) subject may be possible and subject to default permissions defined for the resource. For example:

```
Example ACL: uuid:0000-0000-0000-0000 -> “/oic/*” ? 0x01 (read-only)
```

Details of the format for ACL is defined in section 12.5. Each ACL is composed of one or more ACEs. It is assumed that each OIC device has at least one access control resource. Absence of an ACL on an OIC device is an indication that ACL provisioning may be required and access to the corresponding resource may be denied until the appropriate ACL is provisioned.

It should be noted that the ACL is considered a secure virtual resource and thus 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 OIC device implementations and thus secure storage, usage and isolation of ACLs for different OIC servers on the same device needs to be considered.

5.1.1 ACL Architecture (Informative)

As mentioned, an OIC Client device requests access to resources from an OIC Server. The OIC Server examines the OIC client’s access rights to its resources based on either OIC client’s identity (if SBAC) or role (RBAC). Access requests may be authorized based on group or device credentials. The ACL architecture illustrates four client devices seeking access to server resources. A server evaluates each request using local ACL policies and access manager services.

Each ACE contains the permission set that will be applied for a given resource requestor. Permissions consist of a combination of Create, Read, Update, Delete and Notify (CRUDN) actions. Requestors authenticate as either a device or a device operating with a particular role. OIC 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

OIC servers may host ACL resources locally. Local ACLs allow greater autonomy in access control processing than remote ACL processing by an Access Manager Server (AMS) as described below.

The following use cases intend to describe the operation of access control

Use Case 1: Server device hosts 4 resources (R1, R2, R3 and R4). OIC client device D1 requests access to resource R1 (hosted at OIC 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.

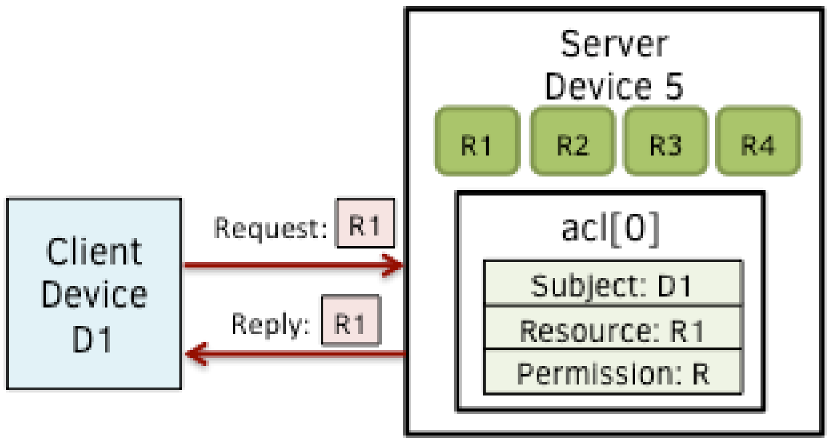


Figure 2 – Use case-1 showing simple ACL enforcement

Use Case 2: OIC client device D2 access is denied because no local ACL match is found for subject D2 pertaining resource R2 and no access manager policy is found.

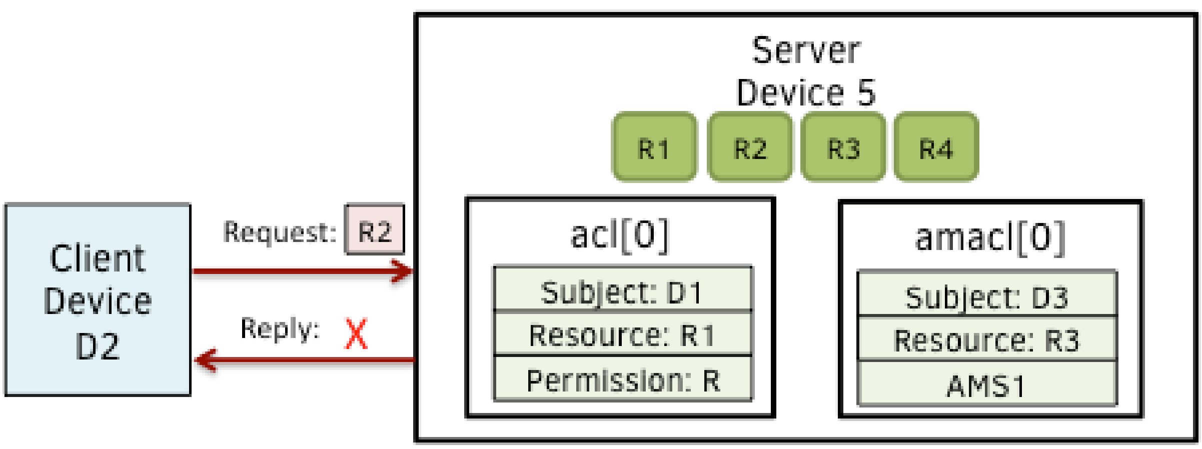


Figure 3 Use case 2: A policy for the requested resource is missing

5.1.1.2 Use of Access Manager Service

Access manager services improve ACL policy management. However, they can become a central point of failure. Due to network latency overhead, ACL processing may be slower.

Access manager services centralizing access control decisions, but OIC 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 access manager service to resolve access decisions. The amacl may be used in concert with local ACLs (/oic/sec/acl).

The provisioning services resource (/oic/sec/svc) shall contain an Access Manager service entry of type oic.sec.ams.

The OIC server device may open a connection to a service of type oic.sec.ams. Alternatively, the OIC server may reject the resource access request with an error that instructs the requestor to obtain a suitable access sacl. The sacl signature may be validated using the credential resource associated with a service of type oic.sec.ams.

Use Case 3: OIC 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

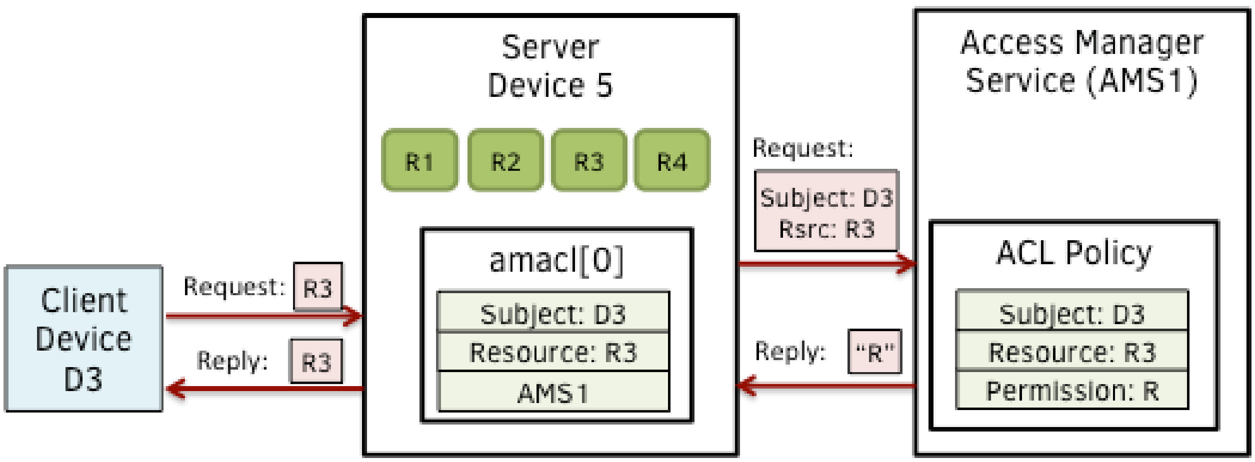


Figure 4 - Use case-3 showing Access Manager Service supported ACL

Use Case 4: OIC 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 an access sacl issuer. Device D4 obtains Sacl1 signed by AMS1 and forwards the SACL to server D5. D5 verifies the sacl signature evaluates the ACL policy that grants Perm2 access.

ACE redirection is that D4 receives an error result with reason code indicating no match exists. D4 reads D4 /oic/sec/svc resource to find who its AMS is then submits a request for a signed ACL. The request is reissued subsequently. D4 is presumed to be known by AMS.

If not, a CMS can be consulted to provision needed credentials.

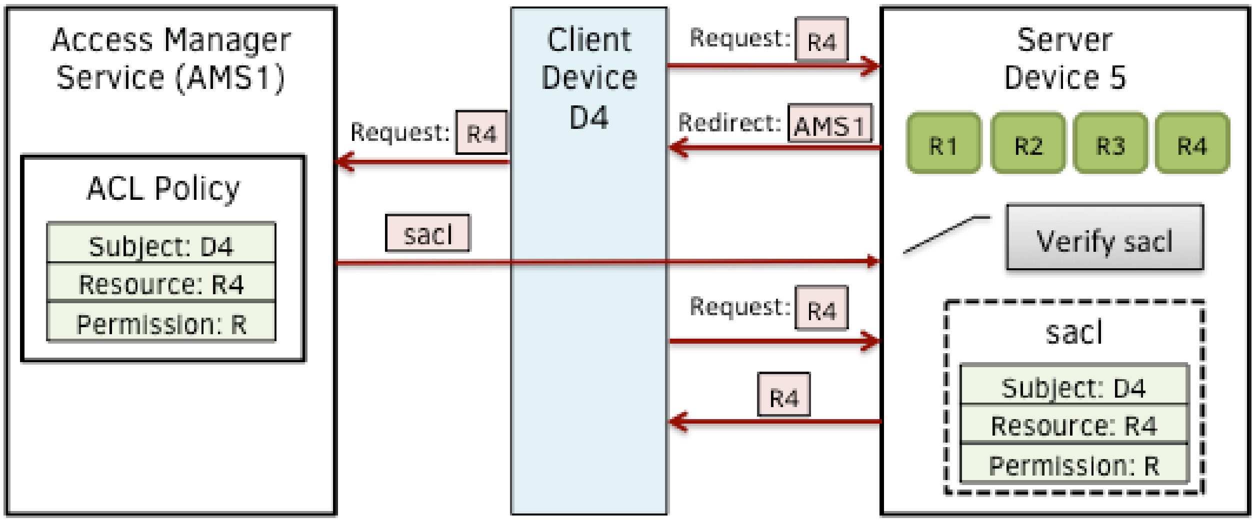


Figure 5 - Use case-4 showing dynamically obtained ACL from an AMS

5.1.2 Access control scoping levels (Informative)

Group Level Access - Group scope means applying AC to the group of OIC devices that are grouped for a specific context. Group credentials may be used when encrypting data to the group or authenticating individual OIC 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.

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

OIC Resource Level Access – OIC Resource level scope means applying AC to individual OIC Resources. Resource access requires an Access Control List (ACL) that specifies how the entity holding the OIC resource (OIC server) shall make a decision on allowing a requesting entity (OIC client) to access the OIC resource.

Property Level Access - Property level scope means applying AC only to a property that is part of a parent OIC resource. This is to provide a finer granularity for AC to OIC resources that may require different permissions for different properties. Property level access control is achieved by creating a Collection resource that references other resources containing a single property. This technique allows the resource level access control mechanisms to be used to enforce property level granularity.

As mentioned, OIC ACL policies are expressed at the resource level granularity. In case, some properties of a resource require different access permissions that the rest of properties within a resource, the resource designer should divide the resource into a collection resource that references the child resources with 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.

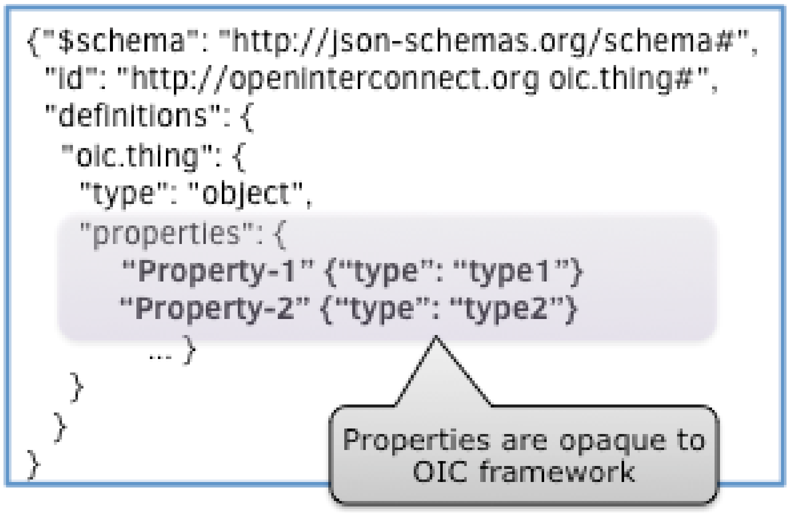


Figure 6 Example resource definition with opaque properties

Currently, OIC 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.

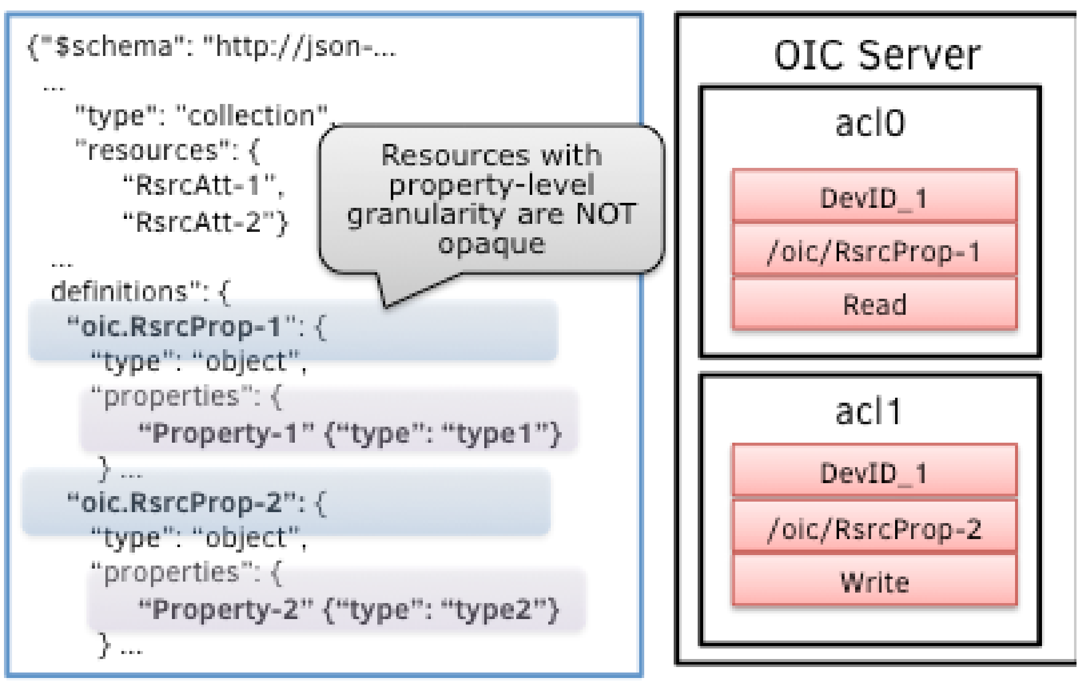


Figure 7 Example resource definition with property-level access control using resource ACLs with Read access for the first property and Write access for the second

5.2 Onboarding and provisioning Overview

In order to provision a new device into the OIC network/ environment, the first step is to onboard the device and perform the necessary security provisioning, which include establishment of ownership as well as creation of identifiers, provisioning of credentials and other security related parameters, needed for secure operation as an OIC device. This section defines the onboarding and security provisioning process but leaves provisioning of other service and application specific parameters to other specifications.

5.2.1 On-Boarding

On-boarding may include a variety of security and non-security related setup to allow a new device to function within the user's OIC network. This may include:

- Configuration of a WiFi access point or other network connectivity setup
- Assignment of an IP address
- Establishing a device owner (or transferring ownership)
- Assignment or registration of a device identifiers (device ID)
- Provisioning of security resources

5.2.2 Establishing a Device Owner

The objective behind establishing device ownership is to establish that a device belongs to a specific IoT network (operated by an owner) where an 'on-boarding' tool (OBT) asserts operational control and management of the device. The process of establishing a device owner includes creation of an ownership context between the new device and the OBT tool. The OBT can be considered a logical entity hosted by any of the tools/ servers mentioned in the following as an example. However, a physical device hosting the OBT will be subject to some security hardening requirements, thus preserving integrity and confidentiality of any credentials being stored. Some examples of tools that could perform the OBT function include 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. For the purposes of this document the tool that establishes device ownership is referred to as the OBT.

Establishing a device owner should be done securely to ensure the device acknowledges it is owned by the intended OBT. This document refers to this process as ownership transfer, since it is assumed that even a new device needs to transfer its ownership from a manufacturer/ seller to a buyer as a new owner. An owner transfer protocol establishes that a new owner (the operator of OBT) is authorized to manage the device. A result of owner transfer is the establishment of the following

- An ownership credential (OC) established at OBT and device. The OC allows the device and OBT to mutually authenticate to each other. OC may be expressed using symmetric or asymmetric cryptography. In this document, the term ownerPSK is used for cases where the ownership credential is a pre-shared symmetric key.
- Creation of device owner transfer method resource (/oic/sec/doxm) that contains a set of properties, including a device identifier associated to the OIC device (logical entity) that is being provisioned within the new device.
- The device needs to know who its owner is. This means the device needs to record the identifier of the OBT (e.g. device ID for OBT). The OBT needs to record the identity of device is part of ownership transfer

- A binding between the device platform ID (if provided by the manufacture) and device ID as a logical identifier, provisioned during ownership transfer.

- Bootstrap information: this is the information as well as credentials needed for the device to interact with the bootstrap server (see next subsection).

This document provides specifications for several alternatives for ownership transfer. Requirements related to implementation of ownership transfer methods are stated in section 7.

As mentioned, part of the ownership transfer is to provision the device with bootstrapping parameters (BP) that allow the device to contact the bootstrap server (BS) and establish a secure session with the BS. The 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.

- Boostrapping credentials (BC): This is the credential that the OIC 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.

As mentioned earlier, the ownership transfer needs to provide the bootstrapping parameters (BP) above. Note that the ownership credentials may be used to provision bootstrapping credentials into the device. For instance if symmetric cryptography is being used as OC and BC, the OC (ownerPSK) may be used in any of the three following methods

- OC can be used as key-encryption key (KEK) for wrapping any BC for the following bootstrapping process
- OC can be used along with PRF to generate bootstrapping keys that are considered child-keys of the OC.
- A symmetric OC may be used as PSK in a PSK-based cipher suite for DTLS authentication.

However, bootstrapping server may also use asymmetric cryptography, such as X.509 certificates for establishing a secure session with the device based upon a pre-existing Trust Anchor, using DTLS and thus may not use OC explicitly in the creation of the BC.

At any rate, the OC should not be used as BC or as credential for any of subsequent network or service provisioning and management activities.

All device owner transfer methods accomplish the following goals:

a. Establish a secure session between new device and the on-boarding tool.
b. Optionally asserts any of the following:
    i. Proximity (using PIN) of the on-boarding tool to the platform.
    ii. Manufacturer’s certificate asserting platform vendor, model and other platform specific attributes.
    iii. Attestation of the platform’s secure execution environment and current configuration status.
    iv. Platform ownership using a digital title.
c. Determines the device identifier.
d. Determines the device owner.
e. Specifies the device owner (e.g. DeviceID of the on-boarding tool).
f. Provisions the device with owner’s credentials.

g. Provisions a 2nd carrier settings and credentials as needed to join the network subsequent to on-boarding and successful owner transfer.

h. Sets the ‘Owned” state of the new device to TRUE.

5.3 Bootstrap process and Security bootstrapping

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, currently assumed to be deployed in the same bootstrap and provisioning tool (BPT)
- Access control policies and ACLs through a ACL provisioning service, currently assumed to be deployed in the same bootstrap and provisioning tool (BPT), but may be part of Access Manager service 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 BPT. Regardless of physical deployment scenario, the same security-hardening requirement (TBD: e.g. protection of credentials used to secure the bootstrapping message exchange with all devices) 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 need to have discovered the bootstrap parameters (BP), including the metadata required to discover and interact with the Bootstrap server (BS) and have been configured with bootstrap credential (BC) required to communicate with BS securely.

In the resource structure, the oic.sec.bss entry in the /oic/sec/svc resource identifies the bootstrap service.

As mentioned, when symmetric keys are used, the ownership credential (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 bootstrap servers (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 a 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 servers, each device may possess an oic.sec.svc resource that describes which service entity to select for provisioning support. To support this, the oic.sec.bss creates or updates the oic.sec.svc resources for

- Credential management service (oic.sec.cms)
- ACL provisioning service (oic.sec.aps)
- Access Manager service (oic.sec.ams)

The idea is that oic.sec.svc resource contains a list of services the device may consult for self-provisioning. Similar to the bootstrapping mechanism, each of the services above must be performed securely and thus require specific credentials to be provisioned. The bootstrap service may initiate of any services above by triggering the device to re-provision its credential resources (oic.sec.cred) for that service.

If symmetric keys are used as credentials for any of the provisioning services above, the bootstrap service needs to provision the appropriate required credentials.

In general, the OIC Server devices may restrict the type of key (CredType) supported.

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 key types; pairwise symmetric keys, group symmetric keys, asymmetric keys and signed asymmetric keys. Keys may be provisioned by a credential management service (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 credential management service. 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 OIC servers, receiving requests for resources they host, need to examine the role asserted by the entity requesting the resource (OIC client) and compare that role with the constraints described in their ACLs. Thus, a OIC client device seeking a role, needs to be provisioned with the required role.

Each OIC device holds the role information as a property within the credential resource. Thus, it is possible that OIC client, seeking a role provisioning, enters a mode where it can provision both credentials and ACLs (if they are provisioned by the same sever!). The provisioning mode/status is typically indicated by the content of /oic/sec/pstat.

Once configured, the OIC client can assert the role it is using by including the role string with the CoAP payload.

e.g. GET /a/light; ‘role’=admin

5.3.5 ACL provisioning

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

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

The ACL provisioning service (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 OIC 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 ACL provisioning service's public key.

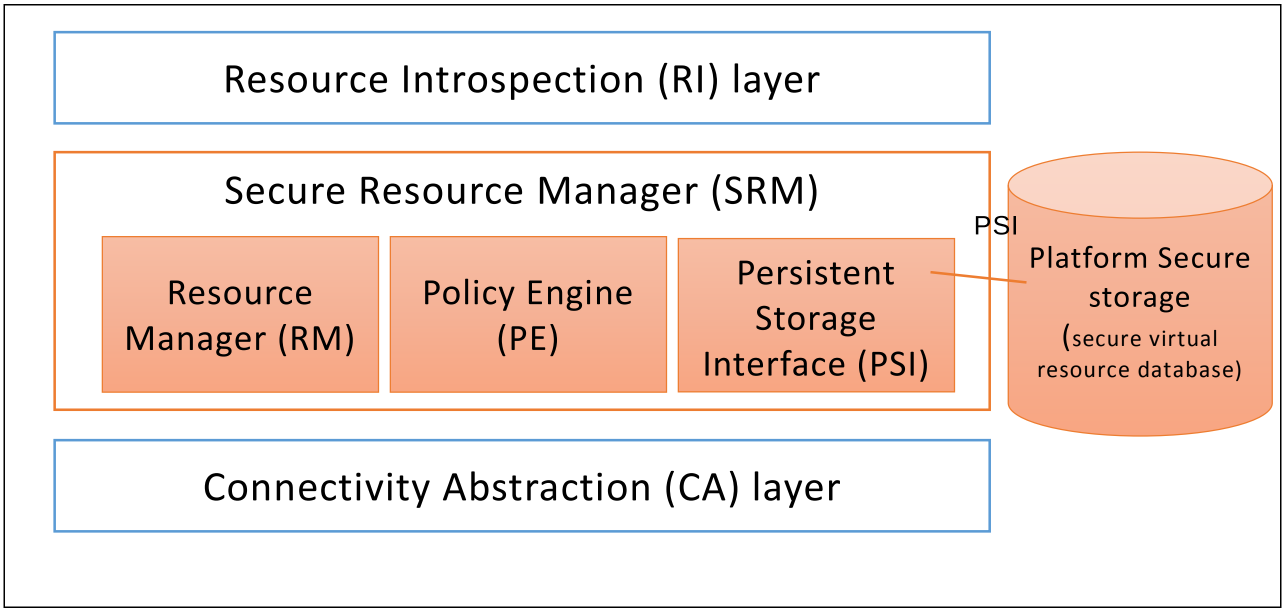
5.4 Secure Resource Manager

Secure Resource Manager (SRM) plays a key role in the overall security operation. In short, SRM performs both management of secure virtual resources (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 Secure Virtual Resources (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 secure virtual resources (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.



5.5 Credential Overview

OIC Devices use credentials to prove the identity of the parties in bidirectional communication. Credentials can be symmetric or asymmetric. Each device stores secret and public (if applicable) parts of it's own credentials, as well as credentials for other devices that have been provided by the On-boarding Tool or a Credential Management Service. These credentials are then used in the establishment of secure communication sessions (e.g. using DTLS) to validate the identities of the participating parties.

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 Core Spec)

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 w/o 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 OIC 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/acl resource contains access control list entries governing access to OIC Server hosted resources. (See Section12.5.2)

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”.

When the discovery request is sent over a non-secure channel (multicast or unicast without DTLS), an OIC Server cannot determine the identity of the requester. In such cases, an OIC 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 OIC client. The OIC 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 ACL 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, an OIC Client with DeviceId “d1” makes a RETRIEVE request on the “/door” Resource hosted on an OIC Server with DeviceId “d3” where d3 has the ACLs below:

```
{
   "Subject": "d1",
   "Resource": "/door",
   "Permission": "00000010", <read>
   "Period": " ",
   "Recurrence": " ",
   "Rowner": "oic.sec.ams"
}
{
   "Subject": "d2",
   "Resource": "/door","Permission": "00000010", <read>
   "Period": " ",
   "Recurrence": " ",
   "Rowner": "oic.sec.ams"
}
{
   "Subject": "d2",
   "Resource": "/door/lock",
   "Permission": "00000100", <update>
   "Period": " ",
```



```
    "Recurrence": " ",
    "Rowner": "oic.sec.ams"
}
{
    "Subject": "d4",
    "Resource": "/door/lock",
    "Permission": "00000100", <update>
    "Period": " ",
    "Recurrence": " ",
    "Rowner": "oic.sec.ams"
}
{
   "Subject": "*",
   "Resource": "/light",
   "Permission": "00000010", <read>
   "Period": " ",
   "Recurrence": " ",
   "Rowner": "oic.sec.ams"
}
```

The ACL indicates that OIC Client “d1” has RETRIEVE permissions on the resource. Hence when device “d1” does a discovery on the /oic/res resource of OIC Server “d3”, the response will include the URI of the “/door” resource. Similarly if an OIC Client “d4” does a discovery on OIC Server “d3”, the response will not include the URI of the “/door” but will include the URI of the “/door/lock” resource. OIC Client “d2” will have access to both the resources.

Discovery results delivered to d1 regarding d3’s /oic/res resource from the secure interface:

```
[
  {
    "d3": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1",
     {
       "href": "/door",
       "rt": "oic.r.door",
       "if": "oic.if.b oic.ll"
     }
  }
]
```

Discovery results delivered to d2 regarding d3’s /oic/res resource from the secure interface:

```
[
  {
```



```
    "d3": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1",
     {
       "href": "/door",
       "rt": "oic.r.door",
       "if": "oic.if.b oic.ll"
     },
     {
       "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:


```
[
  {
    "d3": "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 any device regarding d3's /oic/res resource from the unsecure interface:


```
[
  {
    "d3": "0685B960-736F-46F7-BEC0-9E6CBD61ADC1",
    {
       "href": "/light",
       "rt": "oic.r.light",
       "if": "oic.if.s"
     }
  }
]
```


6.2 Discoverability of security resources

This section will be specified in a future version.

7 Security Provisioning

7.1 Device Identity (Normative)

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

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

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

An OIC 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 more 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 platform in OIC specifications, may host multiple OIC 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 OIC Platform may have 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 separable security.

7.1.1 Device Identity for Devices with UAID

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

For identifying and protecting OIC devices, the platform secure execution environment (SEE) may opt to generate new dynamic public key pair (DPC) for each OIC device it is hosting, or it may opt to simply use the same public key credentials embedded by manufacturer (EPC). In either case, the platform SEE will use its random number generator (RNG) to create a device identity called UAID for each OIC device. The UAID is generated using EPC only or 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 OIC DeviceID is formed from the device's public keys and associated OIC Cipher Suite. The DeviceID is formed by:

1. Determining the OIC 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 OIC device security mechanisms. Use the encoding of the CipherSuite as the 'csid' value in the following calculations. Note that if the OIC Cipher Suite for Dynamic Public key is different from ciphersuite indicated in platform certificate (EPC), OIC Cipher Suite SHALL be used below.

2. From EPC extract the value of embedded public key from a certificate (EPC). 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 opaque 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] # first 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 DeviceID
   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 device new owner. This also allows the ecosystem to extend the trust from manufacturer certificate to a device issued certificate for use of the new DPC and UAID. The degree of trust is in this 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 used in the hash algorithm above to generate UAID.

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

7.2 Device Ownership (Informative)

OIC devices are logical entities that are security endpoints that have an identity that is authenticable using cryptographic credentials. An OIC 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 on-boarding tool (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.

Device ownership transfer logically transitions ownership from a previous owner (e.g. a device manufacturer) to the OBT. Transfer of ownership is achieved through an ad-hoc Diffie-Hellman key exchange.

Ownership transfer protocols should include techniques for establishing the physical proximity of the device to an OBT and establishing the security hardening properties of the device through attestation. Attestation typically requires the use of an embedded manufacturer’s certificate that describes the security properties of the physical platform hosting 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.

7.3 Device Ownership Transfer Methods (Informative)

Device ownership transfer methods facilitate interoperability between devices and on-boarding tools.

The un-owned device does not allow any other function, besides discovery, than to engage in an owner transfer method.

On-boarding typically involves a two stage process for connecting a new device to the owner’s network. During the first step the device connects using a first carrier network that builds an isolated network where only the new device, OBT on optionally provisioning and key management services are reachable.

The owner transfer method is performed establishing ownership credentials. Following successful ownership, the OBT provisions the new device with settings necessary to connect to the regular network via a second carrier network.

The new device restarts to begin the second stage of on-boarding. During the second stage, the new device, now ‘owned’ is discoverable by other devices in the network. The new device however may not be fully provisioned. Provisioning services bring the device to full operational state.

7.3.1 OTM implementation requirements (Normative)

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 OIC, 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. Not withstanding, 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 owner credential 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.

Vendor-specific device owner transfer methods shall adhere to the /oic/sec/doxm resource specification for owner credentials that result 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 *Just-Works* Owner Transfer Method (Normative)

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 OIC resources is a typical step following ownership establishment. The pre-shared key is called OwnerPSK.

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 OIC 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.

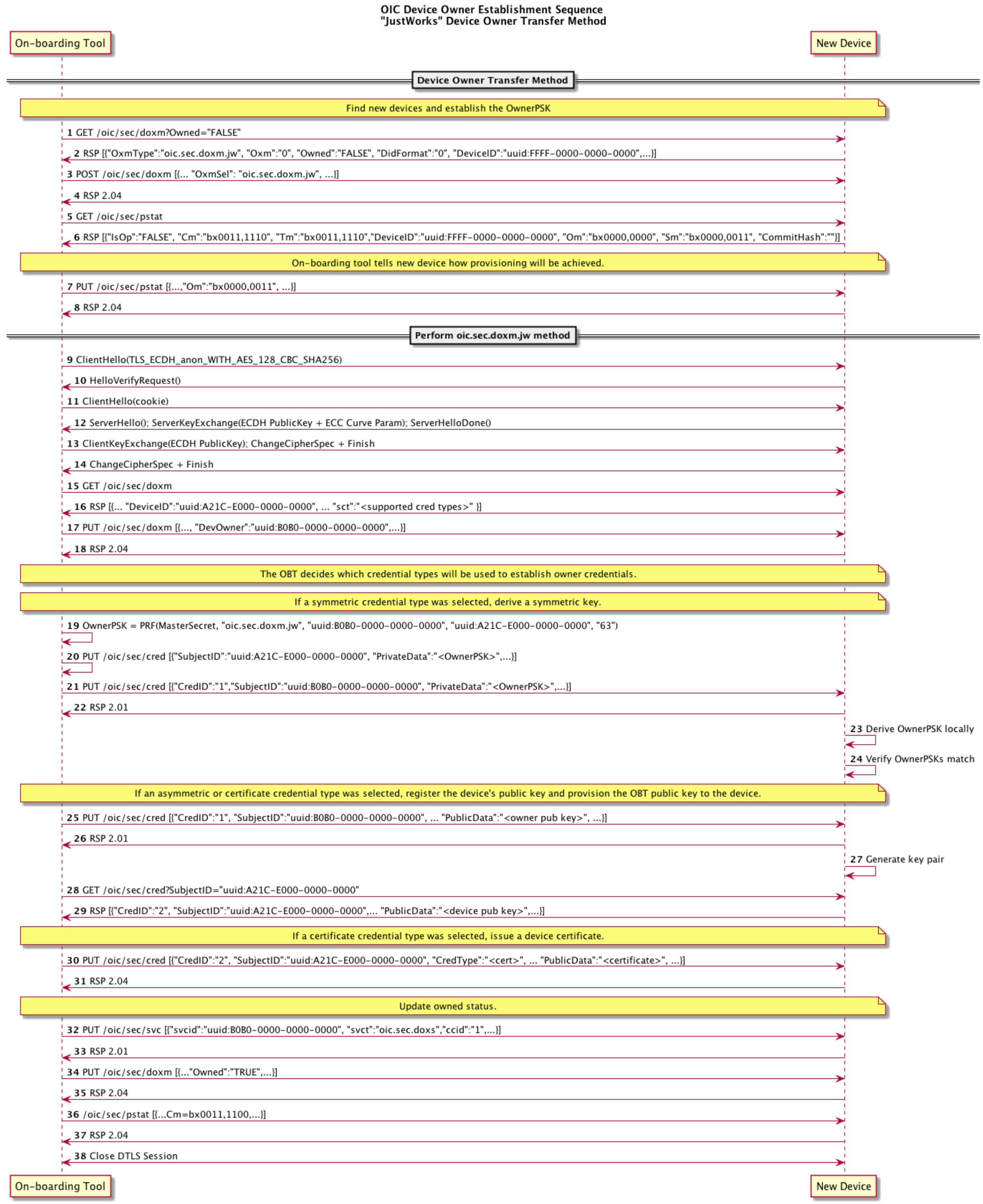


Figure 8 - A 'Just Works' Device Owner Transfer Method

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 should change subsequent to successful owner transfer. The device should supply a temporal ID to facilitate discovery as a guest device.
3, 4	The OBT selects the 'just works' method.
5, 6	The OBT also queries to determine if the device is operationally ready to transfer device ownership.
7, 8	The OBT asserts that it will follow the client provisioning convention.
9 - 14	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.
15, 16	The OBT finds out which credential types the new device can support and decides the ownership credential to provision to the new device.
17, 18	The OBT asserts itself as the owner of the new device and requests device owned status to be changed to TRUE.
19, 20	If symmetric credential type is selected: The OBT uses a pseudo-random-function (PRF) and other information to generate a symmetric key credential - OwnerPSK.
21, 22	If symmetric credential type is selected: The OwnerPSK credential is created on the new device.
23, 24	New device derives the OwnerPSK locally and verifies it matches the value derived by OBT.
25, 26	If asymmetric credential type is selected: The owner public key credential is created on the new device. It may be used subsequently to authenticate the OBT.
27	The new device creates an asymmetric key pair.
28, 29	The OBT reads the new device's asymmetric credential. It may be used subsequently to authenticate the new device.
30, 31	If certificate credential type is selected: Steps 23 – 27 are applied. In addition, the OBT obtains a certificate and instantiates the certificate credential on the new device.
32, 33	OBT creates an entry in the new device's /oic/sec/svc resource that identifies the OBT service.
34, 35	The new device changes the /oic/sec/doxm.Owned status to TRUE and refuses to accept requests to perform ownership transfer methods. The OBT accepts the new device into its database of 'owned' devices.
36, 37	The new device provisioning state is updated.
38	Close the DTLS session.

Table 3 - A 'Just Works' Device Owner Transfer Method Details

7.3.2.1 Just-works Owner Transfer Method Pseudo-Random Function

The OwnerPSK is derived using a PRF that accepts the DTLS MasterSecret value resulting from an anonymous Diffie-Hellman key agreement. The OIC Server and OIC device on-boarding tool shall follow the following format to ensure interoperability across vendor products:

OwnerPSK = *PRF*(MasterSecret, Message, Length);

Where:

- PRF shall use TLS 1.2 PRF defined by RFC5246.
- MasterSecret is the master secret key resulting from the DTLS handshake
- Message is a concatenation of the following:
  - DoxmType string for the just works method (e.g. “oic.sec.doxm.jw”)
  - OwnerID is a URI identifying the device owner identifier and the device that maintains OwnerPSK.
  - DeviceID is new device’s DeviceID (e.g. “urn:uuid:XXXX-XXXX-XXXX-XXXX”).
- Length is the length of Message in octets

7.3.2.2 Security Considerations

Anonymous Diffie-Hellman key agreement is subject to a man-in-the-middle attacker. Use of this method presumes the OBT and the new device perform the ‘just-works’ method assumes on boarding 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.

7.3.3 Random PIN Based Owner Transfer Method

The Random PIN method establishes physical proximity between the new device and the OBT and prevents 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 present the PIN to a Diffie-Hellman key exchange as evidence that someone authorized the transfer of ownership by virtue of having physical access to the new device via the out-of-band-channel.

7.3.3.1 Random PIN Owner Transfer Sequence

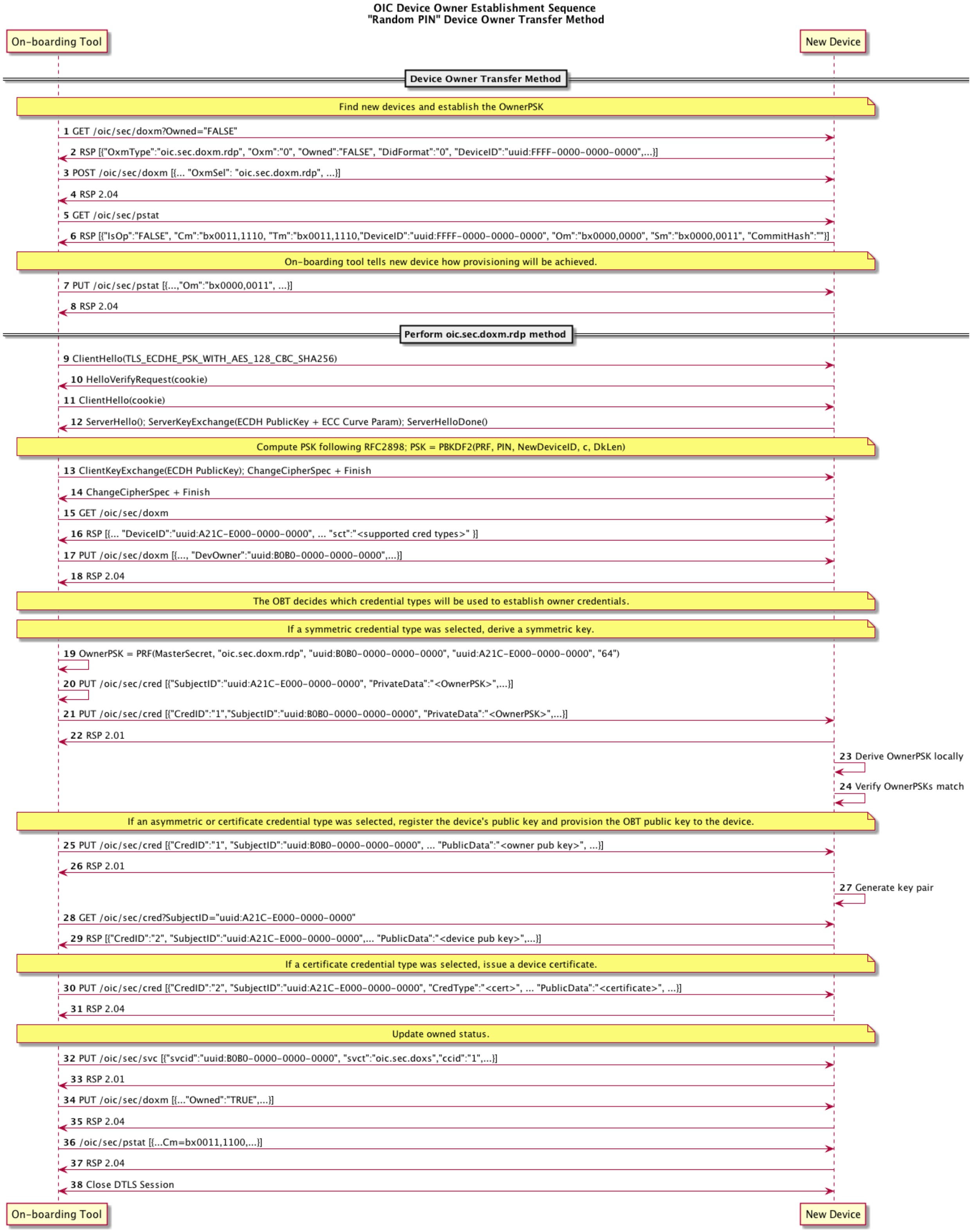


Figure 9 – Random PIN-based Device Owner Transfer Method

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 might change subsequent to successful owner transfer. The device should supply a temporal ID to facilitate discovery as a guest device.
3, 4	The OBT selects the 'Random PIN' method.
5, 6	The OBT also queries to determine if the device is operationally ready to transfer device ownership.
7, 8	The OBT asserts that it will follow the client provisioning convention.
9 - 14	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.
15, 16	The OBT finds out which credential types the new device can support and decides the ownership credential to provision to the new device.
17, 18	The OBT asserts itself as the owner of the new device and requests device owned status to be changed to TRUE.
19, 20	If symmetric credential type is selected: The OBT uses a pseudo-random-function (PRF) and other information to generate a symmetric key credential - OwnerPSK.
21, 22	If symmetric credential type is selected: The OwnerPSK credential is created on the new device.
23, 24	New device derives the OwnerPSK locally and verifies it matches the value derived by OBT.
25, 26	If asymmetric credential type is selected: The owner public key credential is created on the new device. It may be used subsequently to authenticate the OBT.
27	The new device creates an asymmetric key pair.
28, 29	The OBT reads the new device's asymmetric credential. It may be used subsequently to authenticate the new device.
30, 31	If certificate credential type is selected: Steps 23 – 27 are applied. In addition, the OBT obtains a certificate and instantiates the certificate credential on the new device.
32, 33	OBT creates an entry in the new device's /oic/sec/svc resource that identifies the OBT service.
34, 35	The new device changes the /oic/sec/doxm.Owned status to TRUE and refuses to accept requests to perform ownership transfer methods. The OBT accepts the new device into its database of 'owned' devices.
36, 37	The new device provisioning state is updated.
38	Close the DTLS session.

Table 4 - Random PIN-based Device 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 (which is outside the scope this specification) 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, DeviceID, 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.
- DeviceID – UUID of the new device.
- c – Iteration count initialized to 1000, incremented upon each use.
- dkLen – Desired length of the derived PSK in octets.

7.3.3.2 Security Considerations

The Random PIN device owner transfer method security depends on an assumption that the out-of-band method for communicating a randomly generated PIN from the new device to the OBT has not been spoofed.

The PIN value should contain entropy to prevent dictionary attack on the PIN by a man-in-the-middle attacker.

The out-of-band mechanism should be chosen such that it requires proximal context between the OBT and the new device. The attacker is assumed to not have compromised the out-of-band-channel.

OwnerPSK derives additional entropy from the TLS MasterSecret.

7.3.4 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 a signed OBT, which determines the Trust Anchor between the device and the OBT.

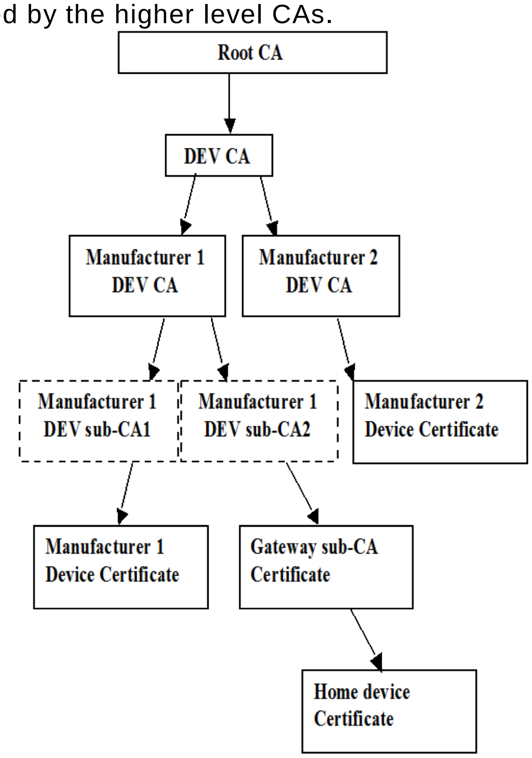
When utilizing certificate-based ownership transfer, devices shall utilize asymmetric keys with certificate data to authenticate their identities with the on-boarding tool ("OBT") in the process of bringing a new device into operation on a user's network. The on-boarding process involves several discrete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

1) Pre-on-board conditions
    a. The device shall be certified by OIC and contain a signed certificate and unique asymmetric key pair
    b. It is recommended that the OBT app binary is signed by a trust anchor/trusted CA to enable mutual authentication with manufacturer-signed clients.
    c. 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 key pair signed by a predetermined trust anchor.
    d. User has configured the OBT app with network access info and account info (if any).
2) Through the OBT, the user connects to the new device using the First Carrier as indicated in Section 7.3
3) Device and OBT shall mutually authenticate using ECDSA to verify each other's signatures. Optionally, the device may bypass OBT authentication by automatically trusting all OBT trust anchors.
4) If authentication fails, the device shall indicate the reason for failure and revert to its pre-on-boarded state.
5) If authentication succeeds, the device and OBT shall establish an encrypted link using ECDH.
6) The OBT shall establish ownership credentials for the device and shall transfer these credentials to the device using the encrypted link.
7) The OBT shall transfer Second Carrier credentials to the device using the encrypted link.

8) Additional ownership transfer provisioning data (e.g. certificates signed by the OBT, user network access information, provisioning functions, shared keys, or Kerberos tickets) may be sent by the OBT to the device.
9) The device shall restart and establish communications to the Second Carrier using credentials received from the OBT. Ownership transfer is now completed.
10) Final state of the device is as follows:
    a. Device shall now be associated with the user network
    b. Device shall no longer accept requests to change ownership
    c. Device shall require credential authentication for any future communication with a new device.
    d. Device may be provisioned with additional credentials for OIC device to device communications. (Credentials may consist of certificates with signatures, UAID based on the device public key, PSK, etc.)

7.3.4.1 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 OIC 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 OSCP 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: C=<county the root created>, O=<name of root CA vendor>, OU=OIC Root CA, CN=OIC (R) Device Root-CA<n>
- DEV CA: C=<country for the DEV CA>, O=<name of root CA vendor>, OU=OIC 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=OIC Manufacturer DEV CA, CN=<name defined by manufacturer><m>
- Device Sub-CA: C=<country device sub-CA>, O=<name of root CA vendor>, OU=OIC 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=OIC Manufacturer OCSP Responder <o>, CN=<name defined by CA vendor >
- Device cert: C=<country>, O=<manufacturer>, OU=OIC Device, CN=<device Type><single space (i.e., " ")><device model name>
    - The following optional naming elements MAY be included between the OU=OIC(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-OIC Device cert, OU=<Gateway UAID>, CN=<device Typle>

Technical Note regarding Gateway Sub-CA: If a manufacturer decides to allow its Gatways 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-OIC Device cert, to indicate that they are not issued by an OIC 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 OIC 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.

The certificate formats below are placeholders and are not finalized in this release of the specification.

7.3.4.2 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 pre-determined Trust Anchor or Certificate Authority. This implies that OIC may need to obtain services from a Certificate Authority (e.g. Symantec, Verisign, etc.) to provide ultimate trust anchors from which all subsequent OIC trust anchors are derived.

The OBT shall authenticate the device. 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 which can compromise network access credentials and/or personal information.

7.3.4.3 Manufacturer’s Certificate Owner Transfer Sequence

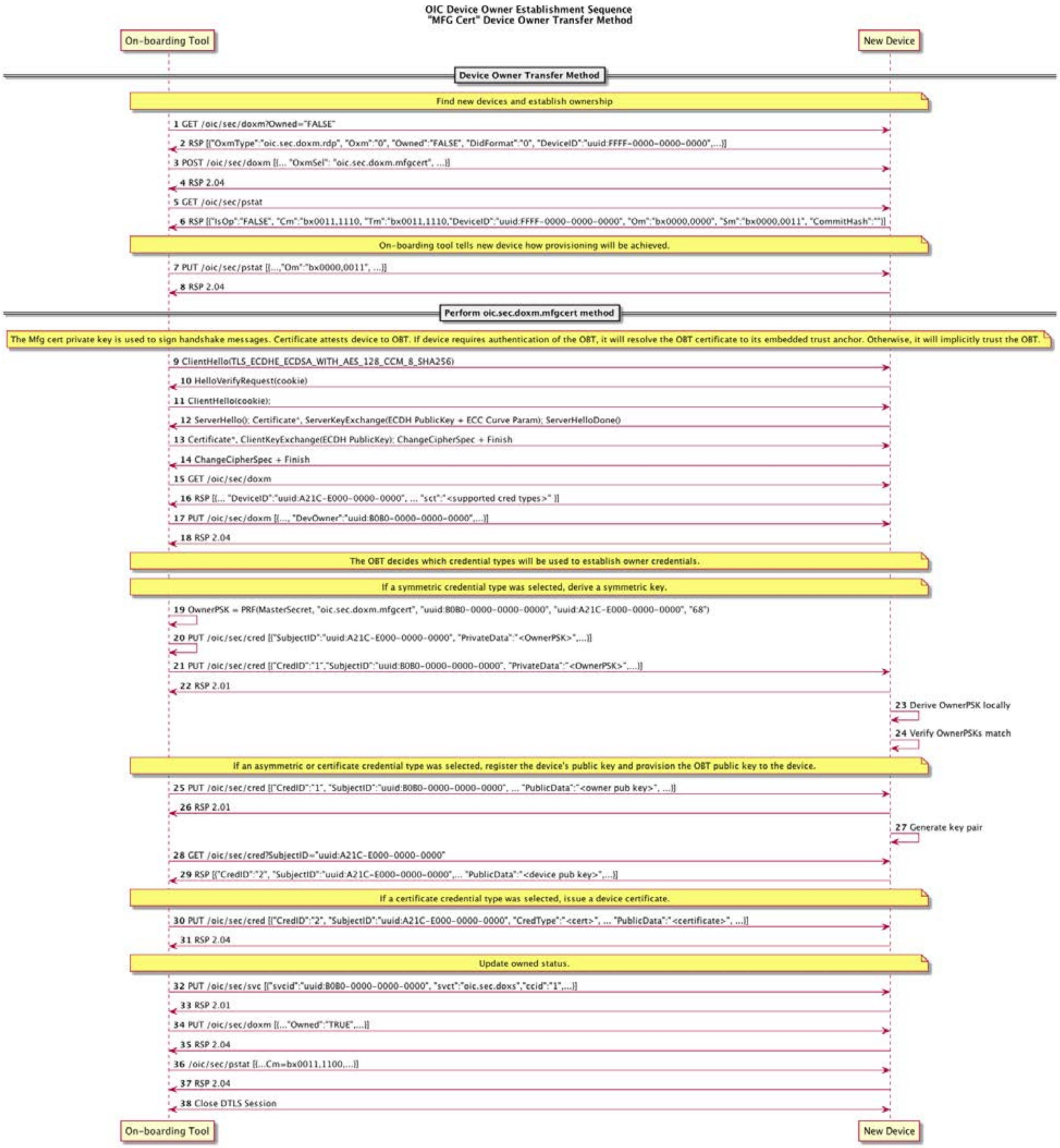


Figure 10 – Manufacturer Certificate Owner Transfer Sequence

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.
3, 4	The OBT selects the ‘Manufacturer Certificate’ method.
5, 6	The OBT also queries to determine if the device is operationally ready to transfer device ownership.
7, 8	The OBT asserts that it will follow the client provisioning convention.
9 - 14	A DTLS session is established using a signed Diffie-Hellman ciphersuite. The manufacturer’s certificate is used to sign the Diffie-Hellman messages. The OBT has been provisioned with the issuer’s trust anchor so that certificate path validation can terminate. If the OBT supplies a Certificate message new device may verify the OBT certificate. The mfg certificate may contain attribute data that describes device hardening and security properties.
15, 16	The OBT finds out which credential types the new device can support and decides the ownership credential to provision to the new device.
17, 18	The OBT asserts itself as the owner of the new device and requests device owned status to be changed to TRUE.
19, 20	If symmetric credential type is selected: The OBT uses a pseudo-random-function (PRF) and other information to generate a symmetric key credential - OwnerPSK.
21, 22	If symmetric credential type is selected: The OwnerPSK credential is created on the new device.
23, 24	New device derives the OwnerPSK locally and verifies it matches the value derived by OBT.
25, 26	If asymmetric credential type is selected: The owner public key credential is created on the new device. It may be used subsequently to authenticate the OBT.
27	The new device creates an asymmetric key pair.
28, 29	The OBT reads the new device’s asymmetric credential. It may be used subsequently to authenticate the new device.
30, 31	If certificate credential type is selected: Steps 23 – 27 are applied. In addition, the OBT obtains a certificate and instantiates the certificate credential on the new device.
32, 33	OBT creates an entry in the new device’s /oic/sec/svc resource that identifies the OBT service.
34, 35	The new device changes the /oic/sec/doxm.Owned status to TRUE and refuses to accept requests to perform ownership transfer methods. The OBT accepts the new device into its database of ‘owned’ devices.
36, 37	The new device provisioning state is updated.
38	Close the DTLS session.

Table 5 - Manufacturer Certificate Owner Transfer Details

7.3.4.4 Security Considerations

The manufacturer certificate private key is embedded in the platform with a high degree of assurance that the private key cannot be copied.

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

The manufacturer certificate defines it’s uniqueness properties.

There may be multiple OIC device instances hosted by a platform containing a single manufacturer certificate

7.3.5 OIC *Decentralized Public Key* (DECAP) Owner Transfer Method

OIC Devices can provide strong authentication using self generated public keys (Referred to dynamically generated credentials, DPC, earlier). The public keys enable a robust and scalable distributed security architecture. The public/private key pairs are also used to derive a unique UAID that can be readily authenticated by peer devices. The generation of OIC Device ID, using DPC is described in an earlier section 7.1. The OIC Device ID is a URI formed from the UAID. The UAID and DeviceId may be shared and used for security management without having to exchange shared secrets. The baseline mechanisms provide support for ACL management without the need for a key distribution center or certificate authority (CA). The use of DECAP does not fully replace the benefits for third party authorization. The use of digital signatures binding properties to the DeviceIds is supported as a means to provide decentralized authorization. As mentioned in section 7.1 for generation of device IDs, embedded certificates and the corresponding credentials (EPC) can, along with DPC, be used in generation of device ID as well as for certification of the self-generated credentials (DPC).

OIC devices, implementing the *DECAP* transfer method shall use the device ID generation mechanism described in section 7.1 to ensure interoperability as extending the trust to the newly generated key pair (DPC). Furthermore, DECAP relies on an authenticated Diffie-Hellman key agreement protocol to arrive at a mutual validation of the peer’s identity and establishment of symmetric keys. The symmetric keys should be used to calculate the Owner Credential.

DECAP may be used to support several models of device on-boarding. The process of introducing one OIC Device to another will vary based on the security requirements and the capabilities for the devices. When a rich UI is available, the UAID may be used as part of the discovery process to act as a ‘secure serial number’ to distinguish similar devices.

7.3.5.1 OIC Device Public Key States

When an OIC Device transitions to the <OOB/whatever name is correct> state it shall generate or derive a new public private key pair. The asymmetric key pair uses the cryptographic parameters and formats determined by the OIC Device Cipher Suite. A DeviceID is formed from the public key and is used for subsequent identification of the device. This Device public/private key should be used to authenticate the OIC Device until the OIC Device transitions to the <reset> state.

When a OIC Device transitions to <Reset>,the public/private key pair shall be deleted and any associated repositories of credentials reset to default values.

7.3.5.2 OIC Cipher Suite

The OIC Cipher Suite determines the format and associated algorithms for a public/private key pair that is established when an OIC Device is first initialized. The OIC Cipher Suites provides the means to prevent cross protocol and cross crypto vulnerabilities by bundling an appropriate set of processing options into a single identifier. An OIC Device should select and support a single OIC Cipher Suite.

The OIC Cipher Suites may be used to support multiple cryptographic options. When multiple OIC Cipher Suites are supported, each option for algorithm support is represented as a different OIC Device with a different OIC DeviceID.

Cipher Suite	Encoding	Suite Parameters
OIC1	0x0101	curve: NIST P256

Cipher Suite	Encoding	Suite Parameters
		hash: SHA256 sign: ECDSA DTLS Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CCM_SHA256 UAID Format: base27
OIC2	0x0102	curve: NIST P521 hash: SHA386 sign: ECDSA DTLS Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CCM_SHA386 UAID Format: base27

7.3.5.3 UAID generation

See section 7.1.1 for UAID generation.

The device public key pair is used during the on-boarding process to create an OwnerPSK using an authenticated key exchange (DTLS based). An out-of-band process should validate the binding of a key pair to a device during the on-boarding process.

The OwnerPSK is the result of an out-of-band transfer of ownership method between the previous owner / manufacturer and the new owner. Both the OOB and Just-Works methods produce a pre-shared key value that is used to assert device ownership. The OwnerPSK must be used to generate the symmetric keys that are used for other purposes. For example, a pair-wise PSK is used to protect device-provisioning data from a system management tool. Easy DECAP may be used to support a simple secure introduction of devices that uses a minimum of out-of-band information.

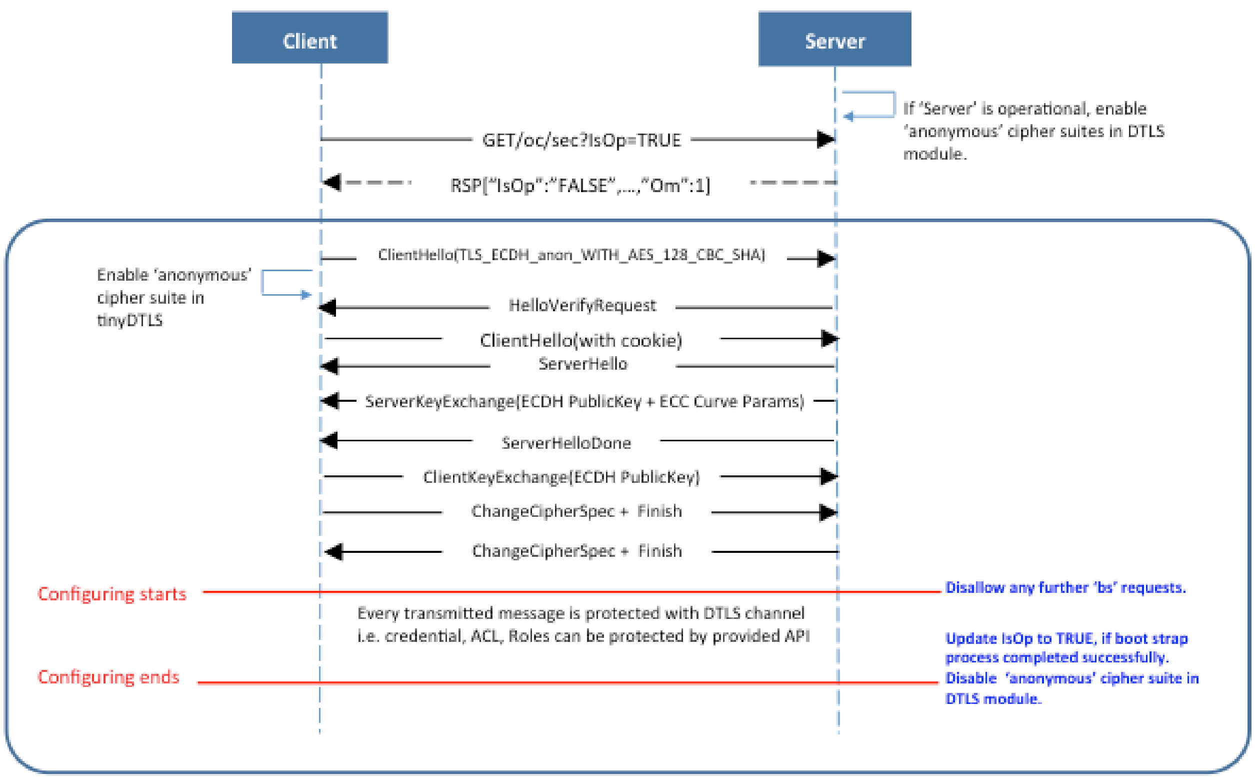


Figure 11 – Easy - DECAP Device Owner Transfer Method

Supported ciphersuites:

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 using RFC 7520
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 using RFC 7520

OwnerPSK = *PRF*(MasterSecret, Message, Length);

Where:

- MasterSecret is the master secret key resulting from the DTLS handshake

- Message is a concatenation of the following:

- DeviceID is the string representation of the newly added device's DeviceID (e.g. urn:uuid:XXXX-XXXX-XXXX-XXXX).

- NewOwnerLabel is string supplied by the owner to distinguish this owner. The new owner must supply this value at device on-boarding. The NewOwnerLabel MAY be a NULL string. For example, the owner's domain name string may be supplied. If the platform contains a platform ownership capability such that multiple OIC device instances hosted on the same platform would not require taking ownership subsequent to the first OIC device instance. The NewOwnerLabel SHOULD identify the platform ownership method and MAY reference the platform owner authorization data. The NewOwnerLabel values may be shared between OIC Device and owner transfer service to facilitate OwnerPSK computation using the prf().

- PrevOwnerLabel is a string supplied by the previous owner that indicates an intention to transfer ownership. The previous owner must supply this value at device on-boarding. He NewOwnerLabel MAY be a NULL string. For example, an owner transfer PIN.

- Length is the length of Message in octets

- PRF MUST use TLS PRF defined by RFC5246.

7.3.6 Vendor Specific Owner Transfer Methods (Normative)

The OIC 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 owner credential(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.

7.3.6.1 Vendor-specific Owner Transfer Sequence Example

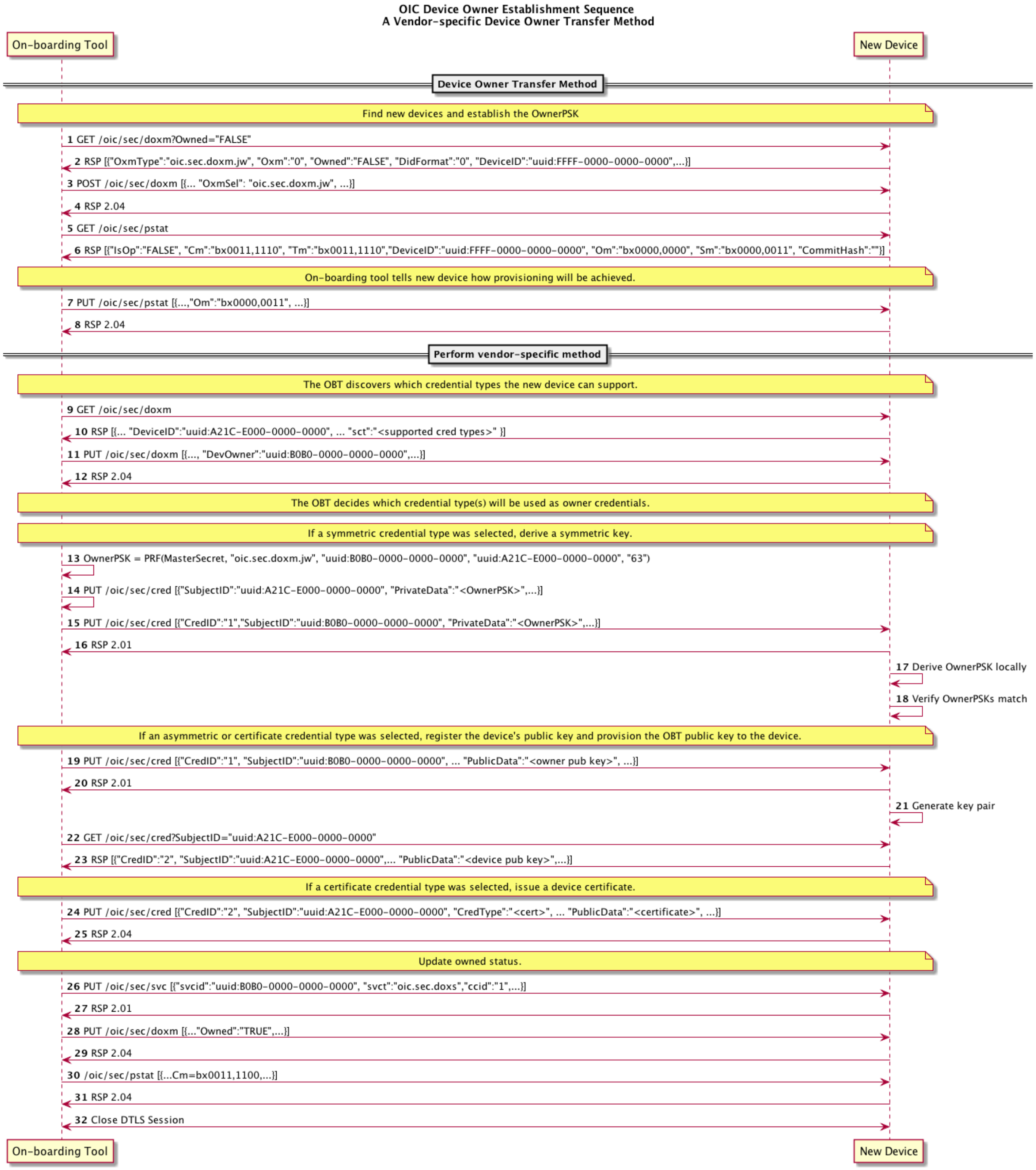


Figure 12 – Vendor-specific Owner Transfer Sequence

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.
3, 4	The OBT selects a vendor-specific owner transfer method.
5, 6	The OBT also queries to determine if the device is operationally ready to transfer device ownership.
7, 8	The OBT asserts that it will follow the client provisioning convention.
9 - 14	The vendor-specific owner transfer method is applied
15, 16	The OBT finds out which credential types the new device can support and decides the ownership credential to provision to the new device.
17, 18	The OBT asserts itself as the owner of the new device and requests device owned status to be changed to TRUE.
19, 20	If symmetric credential type is selected: The OBT uses a pseudo-random-function (PRF) and other information to generate a symmetric key credential - OwnerPSK.
21, 22	If symmetric credential type is selected: The OwnerPSK credential is created on the new device.
23, 24	New device derives the OwnerPSK locally and verifies it matches the value derived by OBT.
25, 26	If asymmetric credential type is selected: The owner public key credential is created on the new device. It may be used subsequently to authenticate the OBT.
27	The new device creates an asymmetric key pair.
28, 29	The OBT reads the new device's asymmetric credential. It may be used subsequently to authenticate the new device.
30, 31	If certificate credential type is selected: Steps 23 – 27 are applied. In addition, the OBT obtains a certificate and instantiates the certificate credential on the new device.
32, 33	OBT creates an entry in the new device's /oic/sec/svc resource that identifies the OBT service.
34, 35	The new device changes the /oic/sec/doxm.Owned status to TRUE and refuses to accept requests to perform ownership transfer methods. The OBT accepts the new device into its database of 'owned' devices.
36, 37	The new device provisioning state is updated.
38	Close the DTLS session.

Table 6 – Vendor-specific Owner Transfer Details

7.3.6.2 Security Considerations

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

7.4 Provisioning

7.4.1 Provisioning Flows

As part of on-boarding a new device a secure channel is formed between the new device and the on-boarding tool. Subsequent to the device ownership status being changed to ‘owned’, there is an opportunity to begin provisioning. The on-boarding tool 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 OIC 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 12.6 for additional detail.

7.4.1.1 Client -directed Provisioning

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

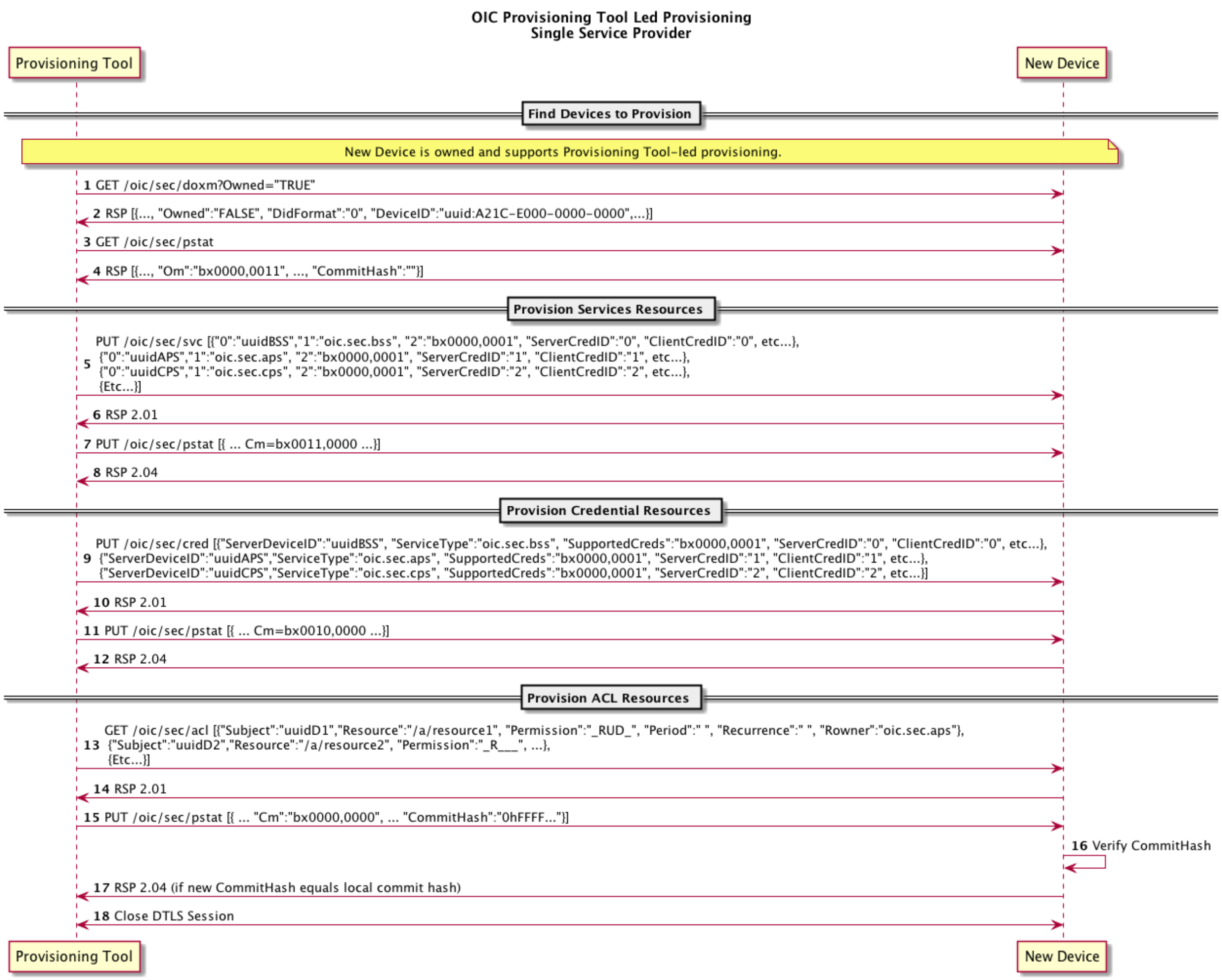


Figure 13 – Example of Client -directed provisioning

Step	Description
1	Discover devices that are owned and support provisioning-tool-led 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 PT-led provisioning, its Om value can be changed.
5 - 6	PT instantiates the /oic/sec/svc resource. The svc resouce includes entries for the bootstrap service, ACL provisioning service and credential management service. It references credentials that should not have been provisioned yet.
7 - 8	The new device provisioning status mode is updated to reflect that various services have been configured.
9 - 10	PT instantiates the /oic/sec/cred resource. It contains credentials for the provisioned services and other OIC devices
11 - 12	The new device provisioning status mode is updated to reflect that the security services have been configured.
13 - 14	PT instantiates /oic/sec/acl resources.
15	The new device provisioning status mode is updated to reflect that ACLs have been configured. The PT computes a hash of all the provisioning messages “CommitHash”. CommtHash is given to the new device.
16	The new device compares the CommitHash with an internal CommitHash value it has computed over the provisioning messages. If these values match, the /oic/sec/pstat.CommitHash property is updated with the new value.
17	The return code reflects successful CommitHash verification and resource update.
18	The secure session is closed.

Table 7 - Steps describing Client -directed provisioning

7.4.1.2 Server -directed Provisioning

Server-directed provisioning relies on the OIC Server (i.e. New Device) for directing much of the provisioning work. As part of the on-boarding process the support services used by the OIC 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.

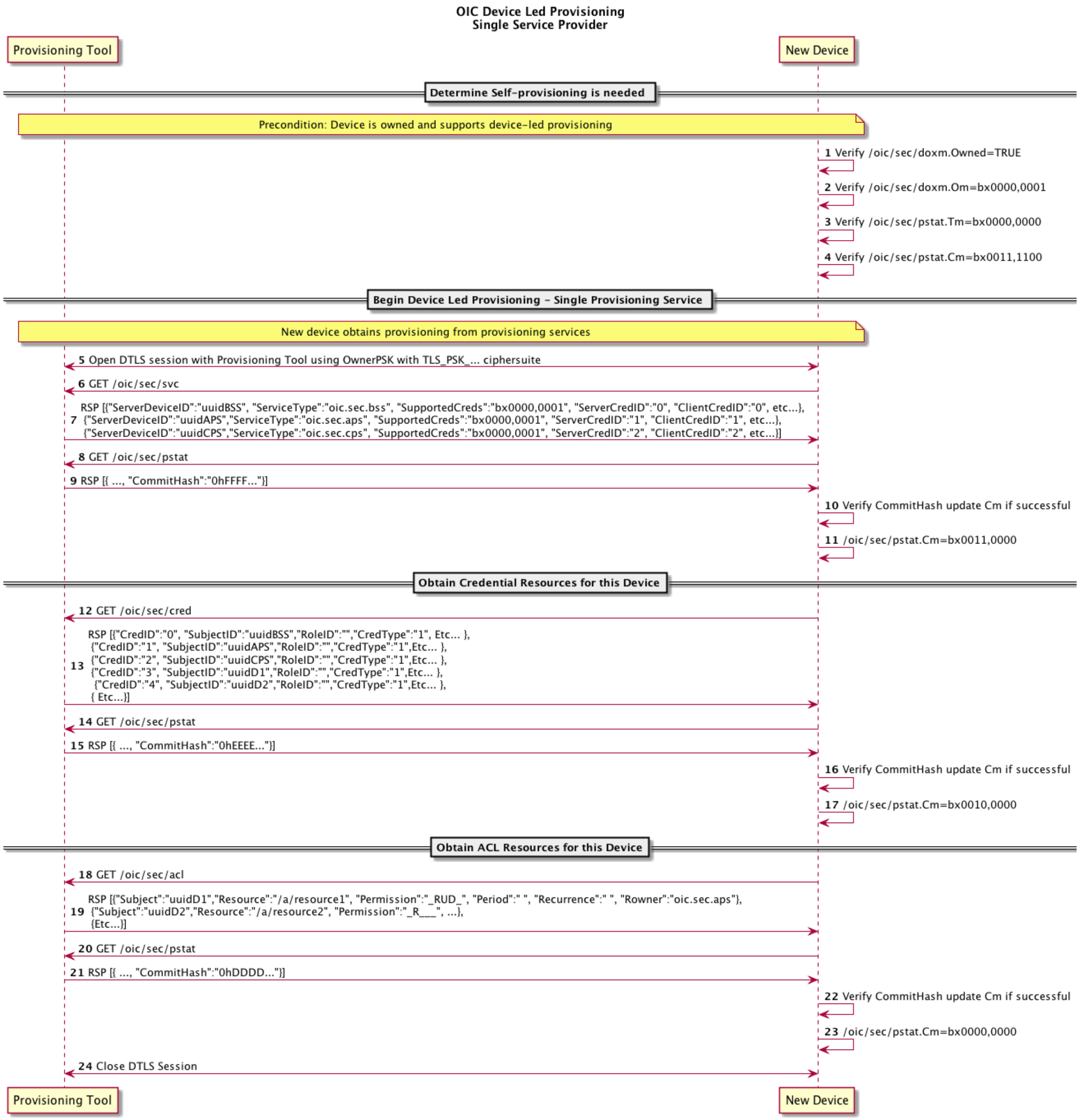


Figure 14: Example of Server-directed provisioning using a single provisioning service

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 OwnerPSK.
6 - 7	The new device gets the /oic/sec/svc resources. The svc resource includes entries for the bootstrap service, ACL provisioning service and credential management service. It references credentials that should not have been provisioned yet.
8 - 9	The new device gets the PT commitHash value.
10	The new device verifies the PT commitHash value matches its local value.
11	The new device updates Cm to reflect provisioning of bootstrap and other services.
12 - 13	The new devices gets the /oic/sec/cred resources. It contains credentials for the provisioned services and other OIC devices.
14 - 15	The new device gets the PT commitHash value.
16	The new device verifies the PT commitHash value matches its local value.
17	The new device updates Cm to reflect provisioning of credential resources.
18 - 19	The new device gets the /oic/sec/acl resources.
20 - 21	The new device gets the PT commitHash value.
22	The new device verifies the PT commitHash value matches its local value.
23	The new device updates Cm to reflect provisioning of ACL resources.
24	The secure session is closed.

Table 8 – 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.

1309

1310 **Figure 15 – 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 OwnerPSK.
6 - 7	The new device gets the /oic/sec/svc resources. The svc resource includes entries for the bootstrap service, ACL provisioning service, ACL management service and credential management service. It references credentials that might not have been provisioned yet.
8 - 9	The new devices gets the /oic/sec/cred resources. It contains credentials for the provisioned services..
10 - 11	The new device gets the PT commitHash value.
12	The new device verifies the PT commitHash value matches its local value.
13	The new device updates Cm to reflect provisioning of support services.
14	The new device closes the DTLS session with the provisioning tool.
15	The new device finds the credential management service (CMS) from the /oic/sec/svc resource and opens a DTLS connection. The new device finds the credential to use from the /oic/sec/cred resource.
16 - 17	The new device requests additional credentials that are needed for interaction with other devices.
18 - 19	The new device gets the CMS commitHash value
20	The new device verifies the CMS commitHash value matches its local value.
21	The new device updates Cm to reflect provisioning of credential resources.
22	The DTLS connection is closed.
23	The new device finds the ACL provisioning and management service from the /oic/sec/svc resource and opens a DTLS connection. The new device finds the credential to use from the /oic/sec/cred resource.
24 - 25	The new device gets ACL resources that it will use to enforce access to local resources.
26 - 27	The new device should get signed ACL resources immediately or in response to a subsequent device resource request.
28 - 29	The new device should also get a list of resources that should consult an Access Manager for making the access control decision.
30 - 31	The new device gets the AMS commitHash value
32	The new device verifies the AMS commitHash value matches its local value.
33	The new device updates Cm to reflect provisioning of ACL resources.
34	The DTLS connection is closed.

Table 9 - Steps for Server-directed provisioning involving multiple support services

7.5 Bootstrap Example

8 Security Credential Management

8.1 Overview

Note, the Core specification doesn't specify that every device shall act as a Server as it pertains to hosting security resources.

8.2 Credential Lifecycle

OIC credential lifecycle has the following phases: (1) creation, (2) deletion, (3) refresh, (4) issuance and (5) revocation. Credential lifecycle may be applied in an ad-hoc fashion using a device owner transfer method or using a guest introduction method or with the aid of a trusted third party such as a credential management service (CMS).

8.2.1 Creation

OIC devices may instantiate credential resources directly using an ad-hoc key exchange method such as Diffie-Hellman. Alternatively, a credential management service (CMS) may be used to provision credential resources to the OIC 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 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 on boarding service.

8.2.2 Deletion

The CMS can delete credential resources or the OIC 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 OIC key management is equivalent to credential suspension.

8.2.3 Refresh

Credential refresh may be performed with the help of a credential management service (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 OIC 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 credential management service (CMS) at regular intervals.

Alternatively, a credential management service (CMS) can be used to refresh or re-issue an expired credential unless no trusted CMS can be found that is recognized by both devices.

If the CMS credential is allowed to expire, the bootstrap service or on boarding service may be used to re-provision the CMS. If the on boarding established credentials are allowed to expire the device will need to be re-on-boarded and re-apply the device owner transfer steps.

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

(Normative) All devices shall support at least one credential refresh method.

8.2.4 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.

8.3 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). (See Section 12.2 for additional details regarding credential types.)

8.3.1 Pair-wise Symmetric Key Credentials

Pair-wise symmetric key credentials have a symmetric key in common with exactly one other peer device. A credential management service (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 OIC device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The OIC device implementer should apply appropriate integrity protection of the /oic/sec/cred resources to prevent unauthorized modifications.

8.3.2 Group Symmetric Key Credentials

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

Group keys do not provide authenticate of OIC devices but only establish membership in a group.

Group keys are distributed with the aid of a credential management service (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 OIC device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The OIC device implementer should apply appropriate integrity protection of the /oic/sec/cred resources to prevent unauthorized modifications.

8.3.3 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.

Asymmetric authentication key pairs are generated by the OIC device and instantiated in the device's /oic/sec/cred resource by the device directly or the key pair is generated by a credential management service (CMS) and provisioned to the device.

The public key is provisioned to a peer OIC device by a credential management service (CMS) or instantiated directly by a peer device using an enrolment protocol that for example requires proof-of-possession.

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 OIC device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The OIC device implementer should apply appropriate integrity protection of the /oic/sec/cred resources to prevent unauthorized modifications.

8.3.4 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 OIC device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The OIC device implementer should apply appropriate integrity protection of the /oic/sec/cred resources to prevent unauthorized modifications.

8.3.5 Certificate Credentials

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

Asymmetric key pair is generated by the OIC device or provisioned by a credential management service (CMS).

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 OIC device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The OIC device implementer should apply appropriate integrity protection of the /oic/sec/cred resources to prevent unauthorized modifications.

8.3.6 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 OIC 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 OIC device implementer should apply hardened key storage techniques that ensure the PrivateData remains private.

The OIC device implementer should apply appropriate integrity protection of the /oic/sec/cred resources to prevent unauthorized modifications.

Note: This should be used for communication between an oic device and a non-OIC device.

8.4 Certificate Based Key Management

8.4.1 Overview

To achieve authentication and transport security during communications in OIC 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 an OIC device is deployed in the OIC network and credential provisioning is supported by a credential management service (Figure 1). 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.

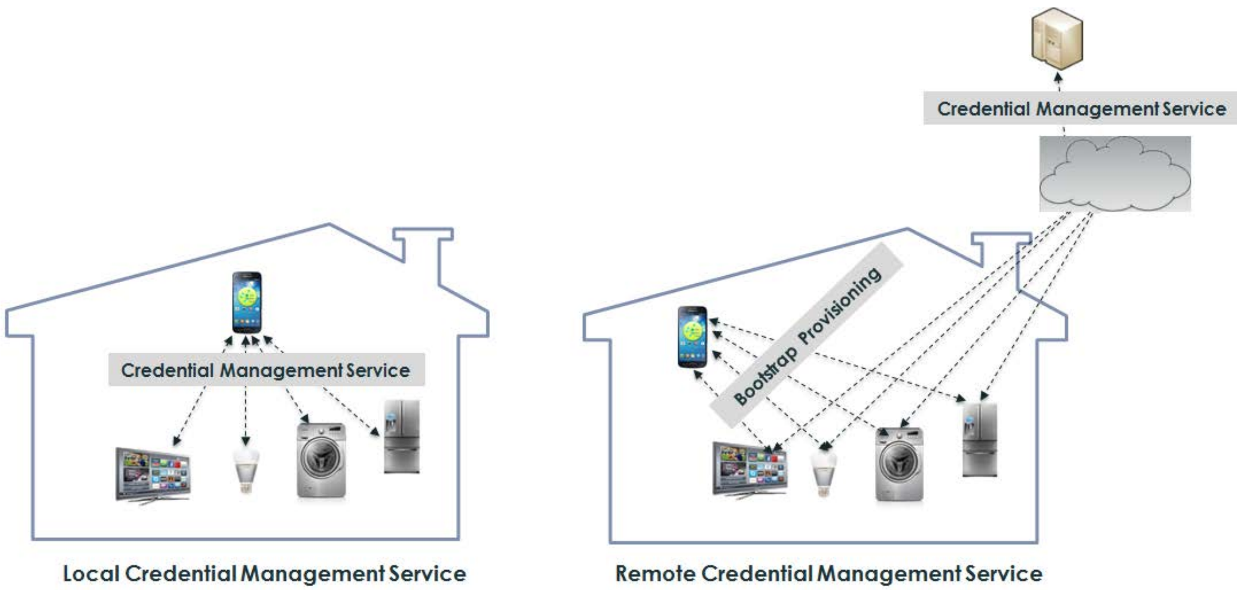


Figure 1 - Certificate Management Architecture

The OIC certificate and OIC 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 is not supported so that the format intends to meet the constrained device's requirement.

As for the certificate and CRL management in the OIC 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 secure resource manager (SRM) is the security enforcement point in an OIC Server as described in Section 5.4, so the data of certificates and CRL will be stored and managed in secure virtual resource database.

The request to issue a device's certificate should be managed by a credential management service when an OIC device is newly on-boarded or the certificate of the OIC 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.

8.4.2 Certificate Format

An OIC certificate format is a subset of X.509 format (version 2 or above) as defined in [RFC5280].

8.4.2.1 Certificate Profile and Fields

The OIC certificate shall support the following fields; `version`, `serialNumber`, `signature`, `issuer`, `validity`, `subject`, `subjectPublicKeyInfo`, `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
- `signatureAlgorithm`: the cryptographic algorithm used by the CA to sign this certificate
- `signatureValue`: the digital signature computed upon the ASN.1 DER encoded `OICtbsCertificate` (this signature value is encoded as a BIT STRING.)

The OIC certificate syntax shall be defined as follows;

```
OICCertificate  ::=  SEQUENCE  {

        OICtbsCertificate      TBSCertificate,
        signatureAlgorithm     AlgorithmIdentifier,
        signatureValue         BIT STRING

}
```

The `OICtbsCertificate` field contains the names of a subject and an issuer, a public key associated with the subject, a validity period, and other associated information

```
 OICtbsCertificate  ::=  SEQUENCE  {
        version            [0]  2 or above,
        serialNumber       CertificateSerialNumber,
        signature          AlgorithmIdentifier,
        issuer             Name,
        validity           Validity,
        subject            Name,
    subjectPublicKeyInfo  SubjectPublicKeyInfo,
}
subjectPublicKeyInfo  ::=  SEQUENCE  {
        algorithm             AlgorithmIdentifier,
        subjectPublicKey      BIT STRING
}
```

The table below shows the comparison between OIC and X.509 certificate fields.

Certificate Fields		Description	OIC	X.509
OICtbsCertificate	version	2 or above	Mandatory	Mandatory
	serialNumber	CertificateSerialNumber	Mandatory	Mandatory
	signature	AlgorithmIdentifier	1.2.840.10045.4.3.2(ECDSA algorithm	Specified in [RFC3279],[RFC

			with SHA256, 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 prime256v1 curve, Mandatory)	Specified in [RFC3279],[RFC 4055], and [RFC4491]
	`issuerUniqueID`	IMPLICIT UniqueIdentifier	Not supported	Optional
	`subjectUniqueID`	IMPLICIT UniqueIdentifier		
	`extensions`	EXPLICIT Extensions		
`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

8.4.2.2 Cipher Suite for Authentication, Confidentiality and Integrity

All OIC 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 OIC 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 prime256v1 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.

8.4.2.3 Encoding of Certificate

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

8.4.3 CRL Format

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

8.4.3.1 CRL Profile and Fields

The OIC 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 `OICtbsCertList` (this signature value is encoded as a BIT STRING.)

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

```
CertificateList  ::=  SEQUENCE  {
       OICtbsCertList        TBSCertList,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING
 }
OICtbsCertList:: = 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
}
```

The table below shows the comparison between OIC and X.509 CRL fields.

CRL fields		Description	OIC	X.509
OICtbsCer	version	Version v2	Not supported	Optional

tList	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

8.4.3.2 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.

8.4.4 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 12.2 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 credential management service (CMS). The OIC 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 OIC device must query the CMS for current revocation status. (See Section 12.3 for additional detail regarding the /oic/sec/crl resource).

8.4.5 Certificate Provisioning

The credential management service (e.g. a hub or a smart phone) issues certificates and private keys for new devices. The credential management service shall have its own certificate and private key pair. The certificate is either self-signed if it acts as Root CA or 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 8.5.2.

The CA in the credential management service shall generate a device's certificate signed by this CA certificate, a paired private key, and then the credential management service transfer them to the device including its CA certificate chain.

The sequence flow of a certificate transfer for a client-driven model is described in Figure 3.

1. The credential management service retrieves information of the device that request a certificate.

2. The credential management service shall transfer the issued certificate, CA chain and private key to the designated device.

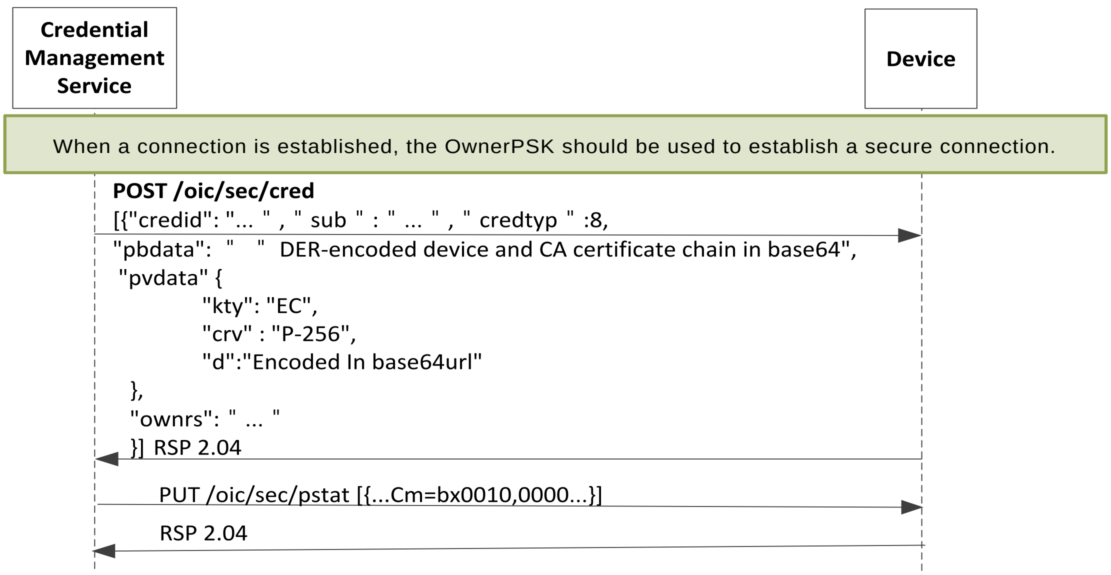


Figure 3 – Client-Driven Certificate Transfer

8.4.6 CRL Provisioning

The only pre-requirement of CRL issuing is that credential management service (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 credential management service 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 OIC network. In some special cases, devices may request CRL to a given credential management service.

There are two options to update and deliver CRL;

- credential management service pushes CRL to each device
- each device periodically requests to update CRL

The sequence flow of a CRL transfer for a client-driven model is described in Figure 4.

1. The credential management service may retrieve the CRL resource property.
2. If the device requests the credential management service to send CRL, it should transfer the latest CRL to the device.

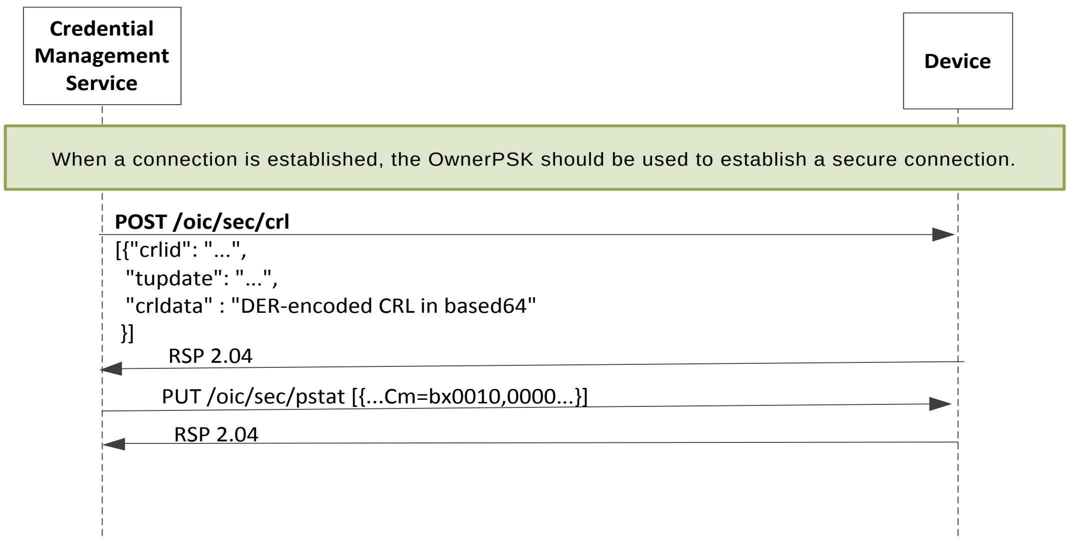


Figure 4 – Client-Driven CRL Transfer

The sequence flow of CRL transferring about server-driven model is described in Figure 5.

1. The device retrieves the CRL resource property tupdate to the credential management service.
2. If the credential management service recognizes the updated CRL information after the designated tupdate time, it may transfer its CRL to the device.

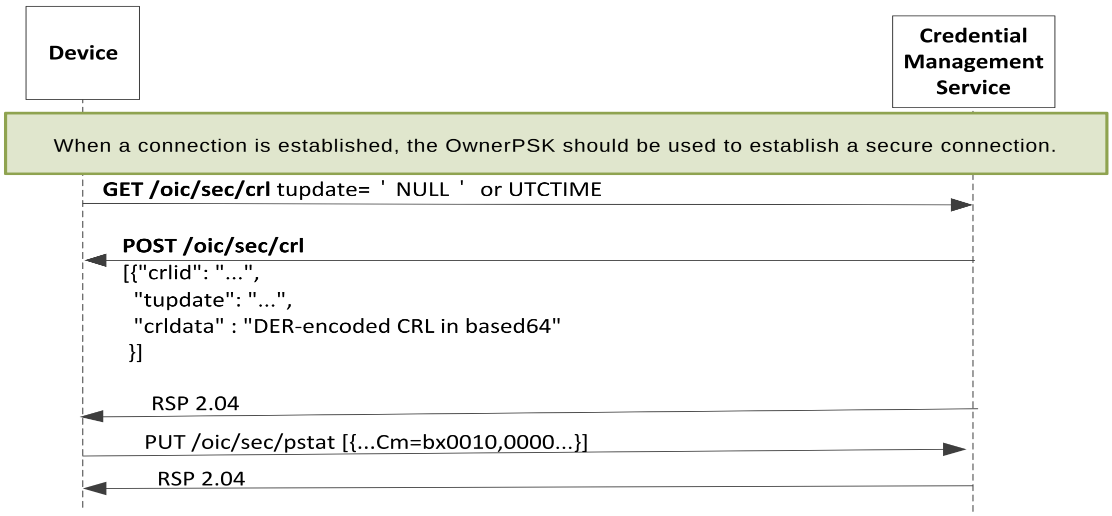


Figure 5 – Server-Driven CRL Transfer

9 Device Authentication

When accessing a restricted resource on an OIC Server, the Server shall authenticate the OIC Client requesting the access. OIC Clients shall authenticate OIC Servers while requesting access.

9.1 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 include 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.

9.2 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.

9.3 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 or its hash is found within the /oic/sec/cred resource. Credential resources found in /oic/sec/cred are used to terminate certificate path validation.

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

10 Message Integrity and Confidentiality

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

10.1 Session Protection with DTLS

OIC Devices shall support DTLS for secured communications as defined in [RFC 6347]. See Section 10.2 for a list of required and optional Cipher Suites for message communication.

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

10.1.1 Unicast Session Semantics

For unicast messages between an OIC Client and an OIC Server, both devices shall authenticate each other. See Section 9 for details on Device Authentication.

Secured unicast messages between a client and a server shall employ an appropriate cipher suite from Section 10.2. The sending device shall encrypt and sign messages as defined by the selected cipher-suite and the receiving device shall verify and decrypt the messages before processing them.

10.1.2 Considerations on Export Licensing with Crypto

10.2 Cipher Suites

Note: Device classes are defined in RFC 7228

10.2.1 Cipher Suites for Device Ownership Transfer

10.2.1.1 Just Works Method Cipher Suites

The oic.sec.doxm.jw owner transfer method may use the following DTLS ciphersuites.

TLS_ECDH_ANON_WITH_AES_128_CBC_SHA256,
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA256,

All OIC devices shall implement:
TLS_ECDH_ANON_WITH_AES_128_CBC_SHA256.
Class-2 and lower devices MAY implement:
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA256
Devices above Class-2 shall implement:
TLS_ECDH_ANON_WITH_AES_128_CBC_SHA256,
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA256

10.2.1.2 Random PIN Method Cipher Suites

The oic.sec.doxm.rdp owner transfer method may use the following DTLS ciphersuites.

TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
TLS_PSK_DHE_WITH_AES_128_CCM_8, (* 8-OCTECT authentication tag *)
TLS_PSK_DHE_WITH_AES_256_CCM_8,
TLS_DHE_PSK_WITH_AES_128_CCM, (* 16-OCTECT authentication tag *)
TLS_DHE_PSK_WITH_AES_256_CCM
Note: All CCM based ciphersuites implement SHA256 integrity value.

See RFC4279, RFC5489 and RFC6655, RFC7251.

All OIC devices shall implement at least one of the following:
TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
TLS_PSK_DHE_WITH_AES_128_CCM_8,

Class-2 and lower devices may implement:

TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
TLS_PSK_DHE_WITH_AES_256_CCM_8,
TLS_DHE_PSK_WITH_AES_128_CCM,
TLS_DHE_PSK_WITH_AES_256_CCM

Devices above Class-2 shall implement:

TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
TLS_PSK_DHE_WITH_AES_256_CCM_8,
TLS_DHE_PSK_WITH_AES_128_CCM,
TLS_DHE_PSK_WITH_AES_256_CCM

10.2.1.3 Certificate Method Cipher Suites

The oic.sec.doxm.mfgcert owner transfer method may use the following DTLS ciphersuites.

TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_SHA256,

Using the following curves:

secp256r21 (See [RFC4492])

See RFC7251.

All OIC devices shall implement at least one of the following:

TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_SHA256,

Class-2 and lower devices may implement:

TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,

Devices above Class-2 shall implement:

TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,

10.2.2 Cipher Suites for Symmetric Keys

The following ciphersuites are defined for DTLS communication using PSKs:

TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
TLS_PSK_DHE_WITH_AES_128_CCM_8, (* 8 OCTET Authentication tag *)
TLS_PSK_DHE_WITH_AES_256_CCM_8,
TLS_DHE_PSK_WITH_AES_128_CCM, (* 16 OCTET Authentication tag *)
TLS_DHE_PSK_WITH_AES_256_CCM,
Note: All CCM based ciphersuites implement SHA256 integrity value.

See RFC4279, RFC5489 and RFC6655.

All OIC devices shall implement:

TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
TLS_PSK_DHE_WITH_AES_128_CCM_8,

Class-2 and lower devices may implement:

TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
TLS_DHE_PSK_WITH_AES_128_CCM,
TLS_DHE_PSK_WITH_AES_256_CCM,

Devices above Class-2 shall implement:

TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256,
TLS_PSK_DHE_WITH_AES_256_CCM_8,
TLS_DHE_PSK_WITH_AES_128_CCM,
TLS_DHE_PSK_WITH_AES_256_CCM,

10.2.3 Cipher Suites for Asymmetric Credentials

The following ciphersuites are defined for DTLS communication with asymmetric keys or certificates:

TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_SHA256,

Using the following curves:

secp256r21 (See [RFC4492])

See RFC7251.

All OIC devices shall implement at least one of the following:
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_SHA256,

Class-2 and lower devices may implement:
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,

Devices above Class-2 shall implement:
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256,

11 Access Control

11.1 ACL Generation and Management

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

11.2 ACL Evaluation and Enforcement (Normative)

The OIC server enforces access control over application resources before exposing them to the requestor. The security manager in the OIC server authenticates the requestor if access is received via the secure port. If the request arrives over the unsecured port, the only ACL policies allowed are for anonymous requestors. If the anonymous ACL policy doesn't name the requested resource access is denied.

A wild card resource identifier should be used to apply a blanket policy for a collection of resources. For example, /a/light/* matches all instances of the light resource.

Evaluation of local ACL resources completes when all ACL resources 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 OIC server opens a secure connection to the Access Manager Service (AMS). If the primary AMS is unavailable, a secondary AMS should be tried. The OIC server queries the AMS supplying the subject and requested resource as filter criteria. The OIC 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 OIC server returns an error. The requester should query the OIC server to discover the configured AMS services. The OIC client should contact the AMS to request an sacl (/oic/sec/sacl) resource. Performing the following operations implement this type of request:

1) OIC client: Open secure connection to AMS.
2) OIC 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) OIC client: POST /oic/sec/sacl [{ …sacl… }]
5) OIC server: verifies sacl signature using AMS credentials and installs the ACL resource if valid.
6) OIC 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.

12 Security Resources (Normative)

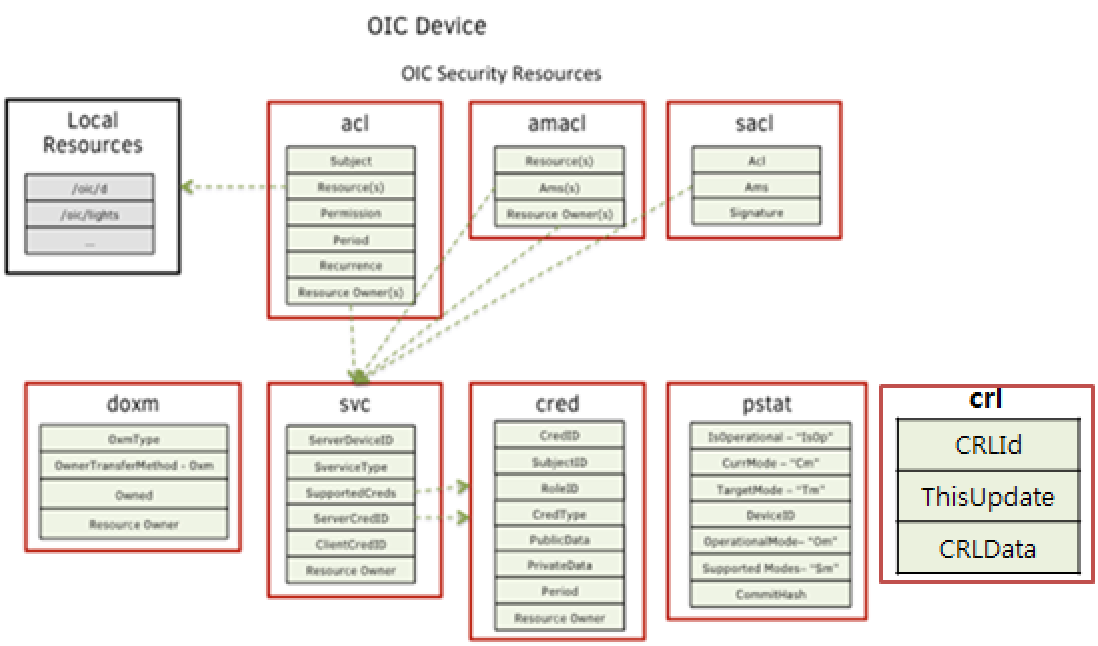


Figure 2 – OIC security resources (`crl` resource is added)

12.1 Device Owner Transfer Resource

The /oic/sec/doxm resource contains the set of supported device owner transfer methods.

Security resource are discoverable through the /oic/res resource. Resource discovery processing respects the CRUDN constraints supplied as part of the security resource definitions contained in this specification.

Owner Transfer Method (OTM) Resource Definition:

Fixed URI	Resource Type Title	Resource Type ID (“rt” value)	Interfaces	Description	Related Functional Interaction
/oic/sec/doxm	Device Owner Transfer Methods	urn:oic.sec.doxm	oic.if.def	Resource for supporting device owner transfer	Configuration

1858 **Table 10 – Owner Transfer Method resource definition**

1859

1860 **Owner Transfer Method Properties Definition:**

Property Title	Property Name	Value Type	Value Rule	Unit	Access Mode	Mandatory	Instance	Description
Owner Transfer Method	Oxm	OxmType	-		R	Yes	Multiple	URN identifying the owner-transfer-method and the organization that defined the method.
Oxm Selection	Oxm Sel	OxmType	-		R	Yes	Single	The Oxm that was selected for device ownership transfer.
SupportedCredential Types	sct	oic.sec.credtype	bitmask		R	Yes	Single	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		R	Yes	Single	Indicates whether or not the device ownership has been established. FALSE indicates device is unowned.
DeviceID Format	DidFormat	UINT8	0-255		R	Yes	Single	Enumerated device ID format. [0 = URN] (e.g. urn:uuid:XXXX-XXXX-XXXX-XXXX)

DeviceID	DeviceID	OCTET[]	-		R	Yes	Single	DeviceID assigned to this instance of the OIC framework. DidFormat determines how to interpret the OCTET string. /doxm.DeviceID informs all other resources containing a device ID including /oic/d. The DeviceID value should not be presumed valid until Owned = True. There can be multiple OIC devices per platform. /oic/p contains a platform identifier that should not be considered as the DeviceID. Refer to the OIC Core specification for more information on /oic/p and /oic/d
Device Owner	DevOwner	oic.sec.svc	-		R	Yes	Single	URI identifying a service that is the device owner. This should be any value chosen by the device owner.
Resource Owner	Rowner	oic.sec.svc	-		R	Yes	Single	This resource's owner. Typically this is the bootstrap service that instantiated this resource

Table 11 **– Owner Transfer Method Properties definition**

The owner transfer method resource contains an ordered list of owner transfer methods where the first entry in the list is the highest priority method and the last entry the lowest priority.

The device manufacturer configures this resource with the most desirable (most secure) methods with high priority and least desirable with low priority. The network management tool queries this list at the time of on boarding 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 Secure Resource Manager (SRM) generates a device identifier (DeviceID) that is stored in the /oic/sec/doxm resource in response to successful ownership transfer.

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

Once owned, the bootstrap service (oic.sec.bss) should change the owned state to ‘0’ (FALSE).

12.1.1.1 OIC defined owner transfer methods

Value Type Name	Value Type URN	Description
OICJustWorks	urn:oic:oic.sec.doxm.jw	The just-works method relies on anonymous Diffie-Hellman key agreement protocol to allow an on-boarding tool to assert ownership of the new device. The first on-boarding tool 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 on-boarding tool and likewise authenticates the on-boarding tool to the device. The device allows the on-boarding tool to take ownership of the device, after which a second attempt to take ownership by a differnet on-boarding tool 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.
OICSharedPin	urn:oic:oic.sec.doxm.rdp	The new device randomly generates a PIN that is communicated via an out-of-band channel to a device on-boarding tool. An in-band Diffie-Hellman key agreement protocol establishes that both endpoints possess the PIN. Possession of the PIN by the on-boarding tool signals the new device that device ownership can be asserted.
OICMfgCert	urn:oic:oic.sec.doxm.mfgcert	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 on-boarding. The manufacturer certificate should contain platform hardening information and other security asserances assertions.

12.2 Credential Resource

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

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

Device Credential Resource Definition:

Fixed URI	Resource Type Title	Resource Type ID (“rt” value)	Interfaces	Description	Related Functional Interaction
/oic/sec/cred	Credentials	urn:oic.sec.cred	oic.if.def	Resource containing credentials for device authentication, verification and data protection	Security

Table 12 – Device Credential resource definition

Device Credential Properties Definition:

Property Title	Property Name	Value Type	Value Rule	Unit	Access Mode	Mandatory	Instance	Description
Credential ID	CredID	UINT16	0 – 64K-1	-	R	Yes	Single	Short credential ID for local references from other resources
Subject ID	SubjectID	oic.uuid	URI	-	R	Yes	Single	Identifies the subject (e.g. device) to which this credential applies.
Role ID	RoleID	oic.sec.role	URI	-	R	No	Multiple	Identifies the role(s) the subject is authorized to assert.
Credential Type	CredType	oic.sec.credtype (UINT16)	One of: [0 | 1 | 2 | 4 | 8 | 16]	-	R	Yes	Single	0: no security mode 1: symmetric pair-wise key 2: symmetric group key 4: asymmetric key 8: certificate 16: PIN /password 32: asymmetric encryption key
Public Data	PublicData	oic.sec.jwk, string OCTET[ ]	-	-	R	No	Single	1:2: ticket, public SKDC values 4, 32: Public key value 8: certificate
Private Data	PrivateData	oic.sec.jwk, oic.sec.tee, String, OCTET[ ]	-	-	-	Conditional Optional	Single	1:2: symmetric key 4: 8, 32: Private asymmetric key 16: password hash, password value, security questions This value shall not be disclosed If the platform hosts a trusted execution environment or secure element then this value should be a handle to the actual object.
Optional Data	Optional Data	OCTET[ ]			R	No	Single	1, 2, 4, 8, 32: revocation status information
Period	Period	String	-	-	R	No	Single	Period as defined by RFC5545. The credential should not be used if the current time is outside the Period window.
Credential Refresh Method	Crm	oic.sec.crm		-	R	No	Single	Credentials with a **Period** property are refreshed using the credential refresh method (crm) according to the type definitions for oic.sec.crm
Resource owner	Rowner	oic.sec.svc			R	Yes	Multiple	Refers to the service resource(s) that should instantiate/update this resource. Rowner status has full (C, R, U, D, N) permission.

1898 **Table 13 - Device Credential Property definition**

All secure device accesses shall have an /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 'Rowner' property.

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

12.2.1 Properties of the Credential Resource

12.2.1.1 Credential ID

Credential ID (CredID) is a local reference to a /oic/sec/cred instance. The Secure Resource Manager (SRM) generates it. CredID shall be used to disambiguate resource instances that have the same SubjectID.

12.2.1.2 Subject ID

Subject ID 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 SubjectID that matches the OIC Server's own DeviceID identifies credentials that authenticate this device.

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

12.2.1.3 Role ID

Role ID identifies the set of roles that have been granted to the SubjectID. 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.

12.2.1.4 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 Public Data, Private Data and Optional Data. 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.

12.2.1.5 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.

12.2.1.6 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 or trusted execution environment 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.

12.2.1.7 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.

12.2.1.8 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 Credential Management Service.

12.2.1.9 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 credential management service initiates re-issuance of credentials nearing expiration. The OIC Server should delete expired credentials to manage storage resources. The Resource Owner property references the provisioning service. The OIC Server uses its /oic/sec/svc resource to identify additional key management service that supports this credential refresh method.
Pre-shared Key	oic.sec.crm.psk	[1]	The OIC Server performs ad-hoc key refresh by initiating a DTLS connection with the OIC 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 OIC Server selects the new validity period. The new validity period value is sent to the OIC Device who updates the validity period for the current credential. The OIC Device acknowledges this update by returning a successful response or denies the update by returning a failure response. The OIC Server uses its /oic/sec/svc resource to identify a key management service that supports this credential refresh method.
Random PIN	oic.sec.crm.rdp	[16]	The OIC 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 OIC 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 OIC Server uses its /oic/sec/svc resource to identify a key management service that supports this credential refresh method.
SKDC	oic.sec.crm.skdc	[1, 2, 4, 32]	The OIC Server issues a request to obtain a ticket for the OIC Device. The OIC Server updates the credential using the information contained in the response to the ticket request. The OIC Server uses its /oic/sec/svc resource to identify the key management service that supports this credential refresh method. The OIC Server uses its /oic/sec/svc resource to identify a key management service that supports this credential refresh method.
PKCS10	oic.sec.crm.pk10	[8]	The OIC Server issues a PKCS#10 certificate request message to obtain a new certificate. The OIC Server uses its /oic/sec/svc resource to identify the key management service that supports this credential refresh method. The OIC Server uses its /oic/sec/svc resource to identify a key management service that supports this credential refresh method.

Table 14 - Credential Refresh Method type definition

12.2.2 Key Formatting

12.2.2.1 Symmetric Key Formatting

Symmetric keys shall have the following format:

128-bit key:

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.

256-bit 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.

12.2.2.2 Asymmetric Keys

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

12.2.2.3 Asymmetric Keys with Certificate

Key formatting is defined by certificate definition.

12.2.2.4 Passwords

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

12.2.3 Credential Refresh Method Details

12.2.3.1.1 Provisioning Service

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

12.2.3.1.2 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”
    - DeviceID_A is the string representation of the device ID that supplied the DTLS ClientHello.
    - DeviceID_B is the device responding to the DTLS ClientHello message
- Length of Message in bytes.

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

12.2.3.1.3 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 OIC Server and communicated to the OIC 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, DeviceID, 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”
- DeviceID – 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 OIC Server and OIC Client use the PPSK to update the /oic/sec/cred resource’s PrivateData property. If OIC Server initiated the credential refresh, it selects the new validity period. The OIC Server sends the chosen validity period to the OIC Client over the newly established DTLS session so it can update it’s corresponding credential resource for the OIC Server.

12.2.3.1.4 SKDC

A DTLS session is opened to the /oic/sec/svc 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 OIC Server updates or instantiates an /oic/sec/cred resource guided by the ticket response contents.

12.2.3.1.5 PKCS10

A DTLS session is opened to the /oic/sec/svc 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 OIC Server. The OIC Server updates or instantiates an /oic/sec/cred resource guided by the certificate contents.

12.2.3.2 Resource Owner

The Resource Owner property allows credential provisioning to occur soon after device on-boarding 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.

12.3 Certificate Revocation List

12.3.1 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.sec.crl		Resource containing CRLs for device certificate revocation	Security

12.3.2 CRL Resource

Property Title	Property Name	Value Type	Value Rule	Unit	Access Mode	Mandatory	Instance	Description
CRL Id	CRLId	UINT16	0 – 64K-1	-	R	Yes	Single	CRL ID for references from other resources
This Update	ThisUpdate	String	-	-	R	Yes	Single	This indicates the time when this CRL has been updated.(UTC)
CRL Data	CRLData	string OCTET[]	-	-	R	No	Single	CRL data based on CertificateList in CRL profile

Technical Note: CRL resource should be defined below for each property.

12.4 Security Services Resource

The /oic/sec/svc resource is used by an OIC device to identify the support services that shall be used to obtain or update security resources. Support services are identified using an OIC DeviceID and require a secure communications channel. The OIC Server and support service shall mutually authenticate. The /oic/sec/svc resource informs the OIC Server regarding which credentials are used to authenticate and verify a given support service. Support services are recognized by a type designation. A support service should implement multiple service types.

Services Resource Definition:

Fixed URI	Resource Type Title	Resource Type ID (“rt” value)	Interfaces	Description	Related Functional Interaction
/oic/sec/svc	Services	urn:oic.sec.svc	oic.if.def	The services resource contains a list of services that are used to configure OIC devices	Configuration

Table 15 – Secure Service resource definition

Security Service Properties Definition:

Property Title	Property Name	Value Type	Value Rule	Unit	Access Mode	Mandatory	Instance	Description
Support Service DeviceID	svcid	oic.uuid		-	R	Yes	Single	Identifies the support service
Service Types	svct	oic.sec.svctype			R	Yes	Multiple	Identifies the type of support implemented by the support service.
Supported Credential Types	sct	oic.sec.credtype	bitmask		R	Yes	Single	Identifies the types of credentials the support service recognizes.
Server Credential ID	scid	UINT16	0 – 64K-1		R	Yes	Single	Local reference to a credential the OIC device uses to authenticate to the support service.
Client Credential ID	ccid	UINT16	0 – 64K-1		R	Yes	Single	Local reference to a credential the OIC device uses to verify the support service.
Credential Refresh Methods	crms	oic.sec.crm			R	No	Multiple	Identifies the credential refresh methods supported by this support service. If the Service Type svt="oic.sec.cms" then crms SHALL be specified.
Resource Owner	rowner	oic.sec.svc			R	Yes	Single	Identifies the support service that can instantiate / update this resource. This refers to an entry in this the /oic/sec/svc resource. This resource shall be instantiated with a resource owner when device ownership is established.

Table 16 - Security Service resource properties definition

Each secure end-to-end connection between an OIC device and its support service shall identify the credentials used to mutually authenticate. A support service should allow multiple authentication methods. The ‘SupportedCreds’ property is used to determine which credential type is appropriate when authenticating to the support service.

Security Service Type Definition:

The security service type oic.sec.svctype defines services that perform device and security management.

Type Name	Type URN	Description
Device Owner Transfer Service	urn:oic.sec.doxs	Service type for (re-)taking ownership of the OIC device into the network
Bootstrap Service	urn:oic.sec.bss	Service type for a bootstrap service that should be used to (re-) provision the /oic/sec/svc resource.
Credential Management Service	urn:oic.sec.cms	Service type for a credential provisioning and management
Access Management Service	urn:oic.sec.ams	Service type for an ACL provisioning and management
Unspecified	urn:*	Service type wildcard that satisfies any service type.

Table 17 – Secure Service type definitions

Support services can proactively seek to establish a secure connection with an OIC device. They inquire as to which support services are supported and have accompanying credentials.

An OIC device identifies acceptable service types used during normal operation by supplying the service type URN.

The asterisk ‘*’ is used when a specific support service type is unspecified.

12.5 ACL Resources

All resources hosted by an OIC 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. DeviceID of the OIC 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 resources accessible via the unsecured communication channel shall be named using the wildcard subject.

12.5.1 OIC Access Control List (ACL) BNF defines ACL structures.

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

<table>
<tr><td><ACL></td><td><ACE>,{<ACE>};</td></tr>
<tr><td><ACE></td><td><SBACE> | <RBACE>;</td></tr>
<tr><td><SBACE></td><td><SubjectId>, <ResourceRef>, <Operation>, [<Validity>,{<Validity>}];</td></tr>
<tr><td><RBACE></td><td><RoleId>,<ResourceRef>,<Operation>,[<Validity>,{<Validity>}];</td></tr>
<tr><td><RoleId></td><td>[<Authority>], ‘/’, [<RoleName>];</td></tr>
<tr><td><RoleName></td><td>[URI]</td></tr>
<tr><td><Authority></td><td>[UUID]</td></tr>
<tr><td><ResourceRef></td><td>[<SSID>] | [<DeviceID>], ’/’, [<ResourceName>,’/’,<Number>]</td></tr>
<tr><td><ResourceName></td><td><URI_String></td></tr>
<tr><td><SubjectId></td><td><DeviceID>, <GroupId>;</td></tr>
<tr><td><SSID></td><td><UInt16></td></tr>
</table>

Figure 16: BNF Definition of OIC ACL

12.5.2 ACL Resource

The /oic/sec/acl resource contains access control list entries governing access to OIC Server hosted resources.

OIC ACL Resource definition:

Fixed URI	Resource Type Title	Resource Type ID (“rt” value)	Interfaces	Description	Related Functional Interaction
/oic/sec/acl	ACL	urn:oic.sec.acl	oic.if.def	Resource for managing access	Security

Table 18 - Local ACL resource definition

OIC ACL Property definition

Row #	Property Name	Opr	Instances	Mandatory	Type	Range	Description
informative	normative	normative	normative	normative	normative	normative	normative
0	Subject	R	Single	Yes	String	-	URN identifying the subjects {Subject} or {Role} who should access {Resource}.
1	Resource(s)	R	Multiple	Yes	String	Fully qualified URI – local URI	URN identifying the resources that have {Permission} rights. NULL matches no resource. Resource path ending in “<path>/*” ‘asterisk’ is a wild card that matching all resource instances at location <path>.
2	Permission	R	Single	Yes	UINT16	0-65535	Access policy in least significant bits. 1st lsb: C(Create), 2nd lsb: R(Read, Observe, Discover), 3rd lsb: U(Write, Update) 4th lsb: D(Delete) 5th lsb: N (Notify)
3	Period	R	Multiple*	No	String	-	Period as defined by RFC5545 *Multiple Period/Recurrence tuple sets.
4	Recurrence	R	Multiple	No	String	-	Recurrence rule as defined by RFC5545
5	Rowner(s)	R	Multiple	Yes	oic.sec.svc	oic.sec.bss, oic.sec.ams	Provisioning service authorized to read, create, update and delete this object.

Table 19 - Local ACL Property definition

Local ACL resources supply policy to a resource access enforcement point within an OIC stack instance. The OIC framework gates OIC client access to OIC 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 identifier of a remote device hosting the resources. Partially qualified references imply 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 an Rowner 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.

12.5.3 Access Manager ACL Resource

Access manager ACL resource definition:

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

Table 20 - Access manager ACL resource definition

Access manager services Property definition:

Row #	Property Name	Opr	Instances	Mandatory	Type	Range	Description
0	Resource(s)	R	Multiple*	Yes	String	-	URN identifying the resource instance to be accessed. (E.g. /oic/d).
1	Ams(s)	R	Multiple*	Yes	oic.sec.svc	oic.sec.ams	The AM service that should issue an access sacl on behalf of the requester. *Multiple AMs are backups in case the primary AM is not available.
2	Rowner(s)	R	Multiple	Yes	oic.sec.svc	oic.sec.bss, oic.sec.ams	Provisioning service authorized to modify this object.

Table 21 - Access manager ACL Property definition

12.5.4 Signed ACL Resource

Signed ACL resource definition:

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

Table 22 – Signed ACL resource definition

Signed ACL property definition:

Row #	Property Name	Operations	Instances	Mandatory	Type	Range	Description
0	Acl	R	Multiple*	Yes	oic.sec.acl	-	A local ACL resource containing an access policy specific to the subject’s

							resource request.
1	Ams	R	Single	Yes	oic.sec.svc	oic.sec.ams	The access sacl issuer 2service.
2	Signature	R	Single	Yes	oic.sec.pk9 oic.sec.jws		Signature bits over the sacl. The signature structure defines the signature format. (e.g. JWS (draft-ietf-jose-json-web-signature-41), PKCS#9 (RFC2985) etc…)

Table 23 – Signed ACL Property definition

12.5.5 Extended ACL Resource

12.6 Provisioning Status Resource

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

Provisioning Status Resource Definition:

Fixed URI	Resource Type Title	Resource Type ID (“rt” value)	Interfaces	Description	Related Functional Interaction
/oic/sec/pstat	Provisioning Status	urn:oic.sec.pstat	oic.if.def	Resource for managing device provisioning status	Configuration

Table 24 – Provisioning Status resource definition

Provisioning Status Properties Definition:

Property Title	Property Name	Value Type	Value Rule	Units	Access Mode	Mandatory	Instance	Description
Is Operational	IsOp	Boolean	T|F	-	R	Yes	Single	Device can function even when Cm is non-zero. Device will only service requests related to satisfying Tm when IsOp is FALSE.
Current Mode	Cm	oic.sec.dpm	0 – 64K-1	-	RW	Yes	Single	Specifies the current device mode.
Target Mode	Tm	oic.sec.dpm	0 – 64K-1	-	RW	No	Single	Specifies a target device mode the device is attempting to enter.
Device ID	DeviceID	urn:oic.uuid	-	-	R	No	Single	Specifies the device to which the provisioning status applies. If not specified, it applies to {this} device.
Operational Mode	Om	oic.sec.dpom	0–255		RW	Yes	Single	Current provisioning services operation mode
Supported Mode	Sm	oic.sec.dpom	0-255		R	Yes	Multiple	Supported provisioning services operation modes
Commit Hash	Ch	oic.sec.sha256	0-UINT256	-	R	Yes	Single	Sha256 hash value of all provisioning commands that have been committed by the device.

Table 25 – Provisioning Status Properties definition

The provisioning status resource /oic/sec/pstat is used to enable OIC 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 /oic/sec/svcs resource to discover whether any suitable provisioning services exist. The OIC device should request provisioning if configured to do so. The /oic/sec/pstat?Om property will specify expected device behavior 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.

The device computes a hash of the CoAP POST or PUT command that was successfully applied by the OIC Server. The OIC Server supplies the current CommitHash property when requesting provisioning; the server extends the hash with the POST or PUT command. If the client fails to commit the POST or PUT, the CommitHash property will not reflect the uncommitted command.

Device Provisioning Mode Type Definition:

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.dpm	Device provisioning mode is a 16-bit bitmask describing various provisioning modes

Device Provisioning Mode Low-Byte:

Value	Device Mode	Description
bx0000,0000 (0)	Normal	Device mode for normal operation
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	Bootstrap service provisioning mode enabling instantiation of a bootstrap service
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)	<Reserved>	Reserved for later use
bx1000,0000 (128)	<Reserved>	Reserved for later use

Device Provisioning Mode High-byte:

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

Device Provisioning Operation Mode Type Definition:

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.dpom	Device provisioning operation mode is a 8-bit bitmask describing various provisioning operation modes

Device Provisioning Operation Mode Bits:

Value	Operation Mode	Description
bx0000,0000 (0)	Multiple devices have different 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.
bx0000,0001 (1)	Single device has all provisioning services	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.
bx0000,0010 (2)	Provisioning service in control of 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.
bx1111,11xx	<Reserved>	Reserved for later use

13 Core Interaction Patterns Security

13.1 Observer

13.2 Subscription/Notification

13.3 Groups

13.4 Publish-subscribe Patterns and Notification

14 Security Hardening Guidelines/ Execution Environment Security

Many TGs in OIC have security considerations for their protocols and environments. These security considerations are addressed through security mechanisms specified in the security specifications for OIC. However, effectiveness of these mechanisms depend on security robustness of the underlying hardware and software platform. This section defines the components required for execution environment security.

14.1 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 (called elements going forward) 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) is 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.

14.1.1 Secure Storage (Informative)

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:

Table 26 Examples of Sensitive Data

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

Exact method of protection for secure storage is implementation specific, but typically a combination of hardware and software methods are used.

14.1.1.1 Hardware secure storage

Hardware secure storage is recommended for use with critical Sensitive Data such as symmetric and asymmetric private keys, access credentials, 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

14.1.1.2 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 14.1.1.1.

14.1.1.3 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.

14.1.2 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.

14.1.3 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).

14.1.4 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).

14.1.5 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 OIC. 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 OIC.

14.1.6 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, OIC can follow that model, if and when OIC engage in distributing sensitive key material (e.g. PKI) to its members.

14.2 Execution Environment security profiles (for discussion)

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 OIC 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 OIC 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 OIC 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

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).

14.2.1.1 Next steps

Define levels of security for each of the security elements for each of the 3 classes.

Define what is needed from each of the elements for secure boot and attestation.

Develop a list of sensitive data for OIC security spec

Develop a list of approved algorithms

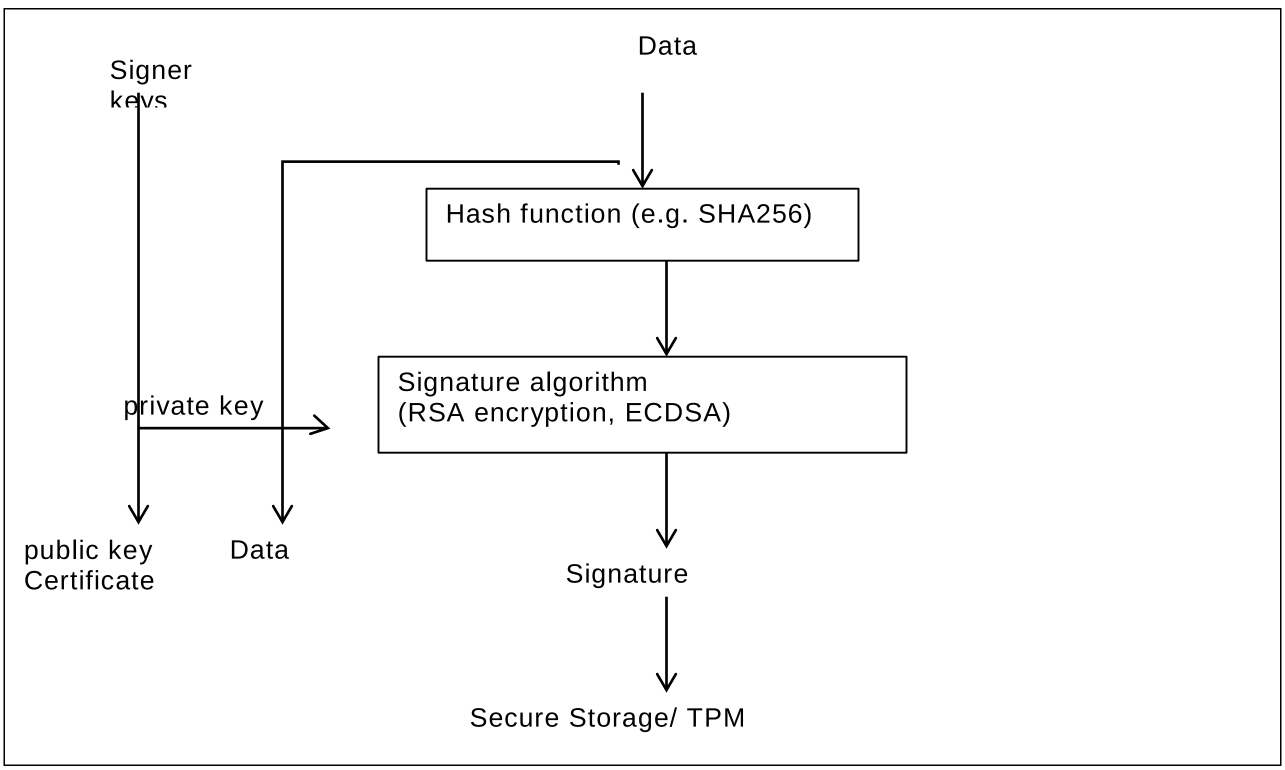
Develop a list of security mechanisms that use time sensitive data (for secure clock)

14.3 Secure Boot

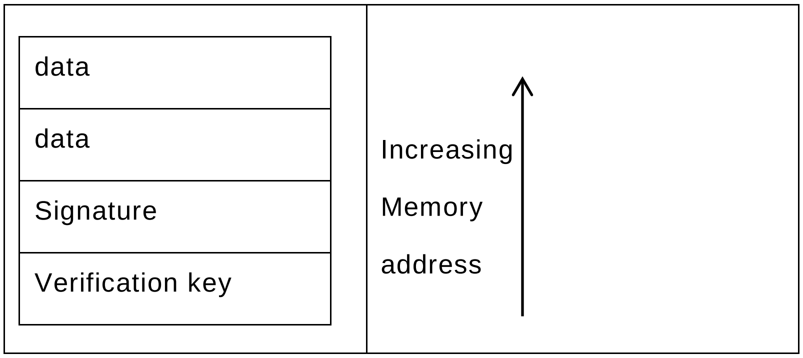
14.3.1 Concept of software module authentication.

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.

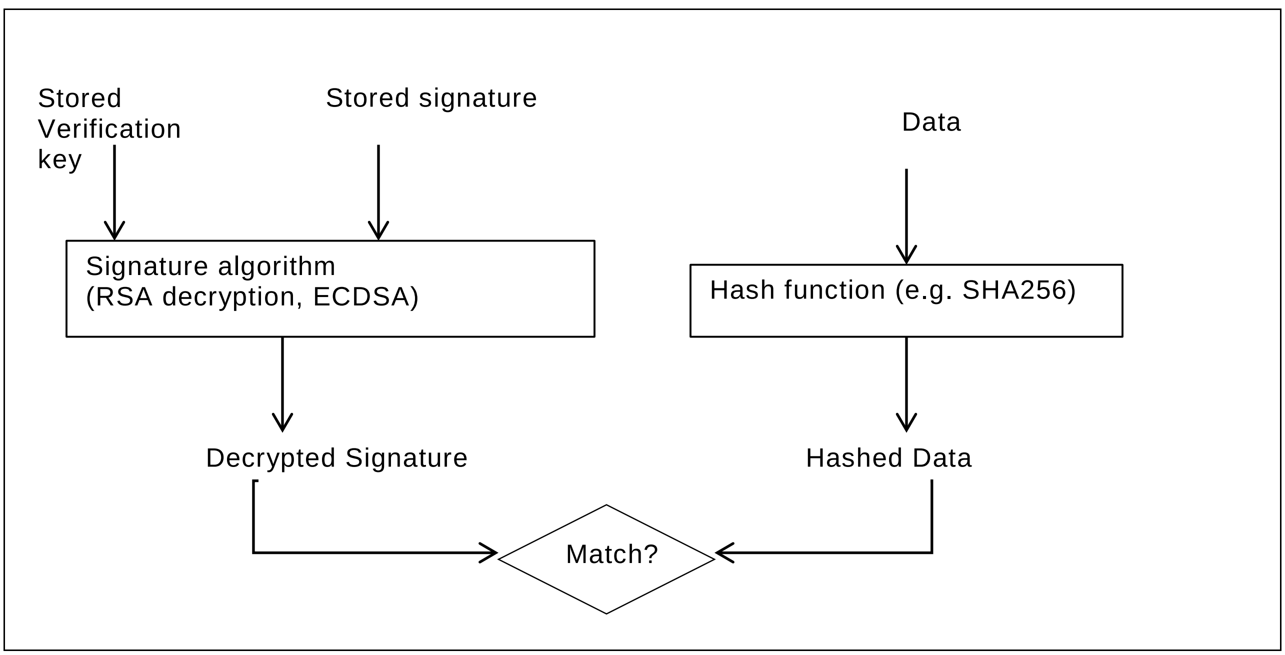
The general concept is the each software module is invoked only after a cryptographic integrity verification is complete. The integrity verification relies on the software module having been hashed (e.g. SHA_1, SHA_256) and then signed with a cryptographic signature algorithm with (e.g. RSA), with a key that only a signing authority has access to.



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.



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.



14.3.2 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.

14.3.3 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.

14.3.3.1 Next steps

Develop a list of approved algorithms and data formats

14.4 Attestation

14.5 Software Update

14.6 Non-OIC Endpoint interoperability

15 Appendix A: Access Control Examples

15.1 Example OIC ACL Resource

The OIC Server is required to verify that any hosted resource has authorized access by the OIC Client requesting access. The /oic/sec/acl 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/acl resource could be configured to enforce access control locally on the OIC Server.

The second local ACL (e.g. /oic/sec/acl/1)

Property Name	Property ID	Property Instance ID	Value	Notes
Subject	0	0	Uuid:XXXX-...-XX01	Subject with ID ...01 should access resources {1,0} and {1,1} with permission {2}
Resource	1	0	{Device1}/oic/sh/light/*	If resource {light, ANY} @ host1 was requested, by subject {0,0}, {0,1} or {0,2} then grant access with permission 0h001F.
Resource	1	1	{Device2}/oic/sh/temp/0	If resource {temp,0} @ host2 was requested, by subject {0,0}, {0,1} or {0,2} then grant access with permission 0h001F.
Permission	2	-	0h001F	C,R,U,D,N permission is granted
Period	3	0	20150101T180000Z/20150102T070000Z	The period starting at 18:00:00 UTC, on January 1, 2015 and ending at 07:00:00 UTC on January 2, 2015
Recurrence	4	0	RRULE:FREQ=WEEKLY;UNTIL=20150131T070000Z	Repeats the {period} every week until the last day of Jan. 2015.
Rowner	5	0	oic.sec.svc?rt="oic.sec.ams"	An ACL provisioning and management service should be identified as the resource owner.

Table 27 - Example acl resource

15.2 Example Access Manager Service

The Access Manager Service (AMS) should be used to centralize management of access policy, but requires OIC 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.

Access Manager Service Resource (e.g. /oic/sec/amacl/0)

Property Name	Property ID	Property Instance ID	Value	Notes
Resource	0	0	{Device1}/oic/sh/light/*	If the {Subject} wants to access the /oic/sh/light/* resources at host1 and an AM sacl was supplied then use the {1} sacl validation credential to enforce access.
Resource	0	1	{Device2}/oma/3	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.
Resource	0	2	/*	If the {Subject} wants to access any local resource and an AM sacl was supplied then use the {1} sacl validation credential to enforce access.
OIC Access manager	1	0	href://<address>/oic/sec/am/0	Forwarding reference for where requestor should obtain a signed ACL.
OIC Access manager	1	1	href://<address>/oic/sec/am/1	Secondary forwarding reference for where requestor should obtain a signed ACL.
Rowner	2	0	oic.sec.svc?rt="oic.sec.ams"	An ACL provisioning and management service should be identified as the resource owner.

Table 28 - Example access manager resource