
# Credential Management for Internet of Things Devices

Internet Protocol for Smart Objects (IPSO) Alliance

Editors:

Hannes Tschofenig, ARM Limited

Ned Smith, Intel

Contributors:

Mark Baugher, Consultant

Per Ståhl, Ericsson

Alan Grau, Icon Labs

Jerker Delsing, Lulea University of Technology

December 2017

# Introduction

Connecting devices to the Internet provides a range of advantages, including remote control, monitoring, fault diagnosis and the ability to collect data for analysis. It may seem that connecting devices equipped with modern microcontrollers is a trivial task with today's Internet protocols. Unfortunately, this is far from true since the pure data transport is only one part of the story. In fact, the pure data transport (without security) is frequently the goal of hackathons and other hands-on IoT workshops. Embedded development with the available libraries, real-time operating systems and IDEs has made it easy to build prototypes within hours.

To interact with such IoT devices securely, the communication protocol between the device and any other communication partners needs integrity and confidentiality protection, otherwise messages can be modified in transit or eavesdropped. To prevent man-in-the-middle attacks, authentication is required between the communication endpoints. There are various communication patterns, as described in [1] and [2]. This whitepaper explores the Device-to-Cloud Communication Pattern (as described in Section 2.2 of [2]) since it illustrates the use of IoT devices well and is frequently used in today's deployments.

The three security services (authentication, confidentiality and integrity) counter common security vulnerabilities found in today's IoT devices, particularly when provided by state-of-the-art security protocols. However, one big challenge remains: *credentials* have to be available on these devices for any communication security protocol to provide their service. Consequently, this leads to an additional requirement, to securely provision credentials to the IoT device. A credential typically consists of keying material, algorithm specific parameters, and a list of entities the credentials can be used with. Each credential also has an identifier associated with it and a lifetime.

Information stored with a credential varies with the type of credential, for example, an asymmetric credential also requires trust anchors [3] to be stored with the credential. A credential is unique to a specific device, and to enhance privacy, there may also be additional credentials for different services, offering unlinkability properties. ”This also lowers the risk that the exposure of a credential may open an attack against a number of services. Sharing the same credential across a number of devices or an entire product family is a security oversight, since it allows an adversary to mount an offline, physical attack on a device and then use it in a remote, online attack. Such attacks are beneficial to the adversary since they can scale to a large number of devices with little additional overhead; refer to [4] for an example. Note that the details of such an attack vary with the type of credential being used. Table 1 shows three types of credentials that are commonly used. Often, security protocols combine different credential types for performance reasons. For example, the Datagram Transport Layer Security (DTLS) handshake may use a certificate (as a device credential) and turn it into short lived pre-shared secret credentials (as a session ticket) as part of the handshake.

Table 1: Credential Types.

| | **Pre-shared keys (PSK)** | **Raw public keys (RPK)** | **Certificates** |
|---|---|---|---|
| **Message Size** | Small | Medium<br>(for transfer of raw public key) | Large<br>(dependent on the length of the certificate chain) |
| **Cryptographic Overhead** | Minimal cryptographic overhead | Large (for Diffie-Hellman and signature operations) | Large (for Diffie-Hellman and signature operations) |
| **RAM Requirements** | Low | Large<br>(for asymmetric crypto computations) | Largest<br>(due to the additional certificate processing) |
| **Code Size Requirements** | Minimal | Medium since it still requires a bignum library. | Large due to the requirement for a bignum library and ASN.1 code for certificate and certificate chain parsing. |

Note: The use of passwords in IoT devices is discouraged due to their low entropy. Passwords are only used in combination with a strong password-based authenticated key exchange protocol. Hence, passwords are not listed in this table. There are also credential types defined that combine PSKs with public key cryptography either in form of strong-password-based authenticated key exchanges, such as the Secure Remote Password (SRP) protocol or with the use of PSK augmented Diffie-Hellman exchanges. We do not consider those in this whitepaper since the main benefits of PSKs become void.

A single IoT device typically stores multiple credentials since there may be a number of credentials in use from different parties, established during the phases of the device lifecycle, such as:

1. Manufacturer credentials: credentials provisioned during manufacturing,
2. operational credentials: credentials established during commissioning, first-time use, ownership change, service change,etc.

Note that credential provisioning during manufacturing covers provisioning during all phases of manufacturing (from the chip, to module and finally device) and includes the system integration phase that typically follows manufacturing. Note also, however, that the term *manufacturer credential* does not require the device to connect to the manufacturer. Instead, the manufacturer credentials on the device, for example, can be used within an enterprise for device local communication.

Once these credentials are available on the IoT device, they can be used as input to a communication security protocol, such as DTLS. The device can then securely communicate with the parties the credentials are intended for.

A device may need more than one credential, and not all credentials can be provisioned to the device during manufacturing. For example, a device may need credentials to connect to a cloud-based server to report sensor readings, but it also needs credentials for accessing the local WiFi, or to utilize a cellular connection in order to access the Internet. The type of credentials used for connectivity heavily depends on the underlying technology, but this is not further discussed in this document. Instead, this whitepaper focusses on the IoT credentials needed to securely access application services, such as cloud services for uploading sensor readings, for remote maintenance of the IoT device, or for providing software updates.

Note that credentials generated during manufacturing may, in some cases, only be used for commissioning and are then replaced by other long-lived credentials with similar role. Such a credential may be generated by the device following commissioning or by the device owner. The exchange of manufacturer credentials is not discussed further in this whitepaper.

Unlike manufacturer credentials, which may last for the lifetime of the physical device, operational credentials are typically created with a shorter lifetime. To distribute these operational credentials to IoT devices, a new entity is introduced, called the key distribution center (KDC). Other terms instead of KDC are used in standards, as illustrated with the example protocols below. The KDC has several functions, such as:

- authenticating IoT devices that connect to it,
- making sure that the devices are authorized to receive the appropriate information,
- distributing operational credentials to devices,
- providing an appropriate user interface to administrators or even end users.

The introduction of the KDC also allows IoT devices to be managed with ease and at scale. An architecture that includes the KDC is shown in Figure 1.

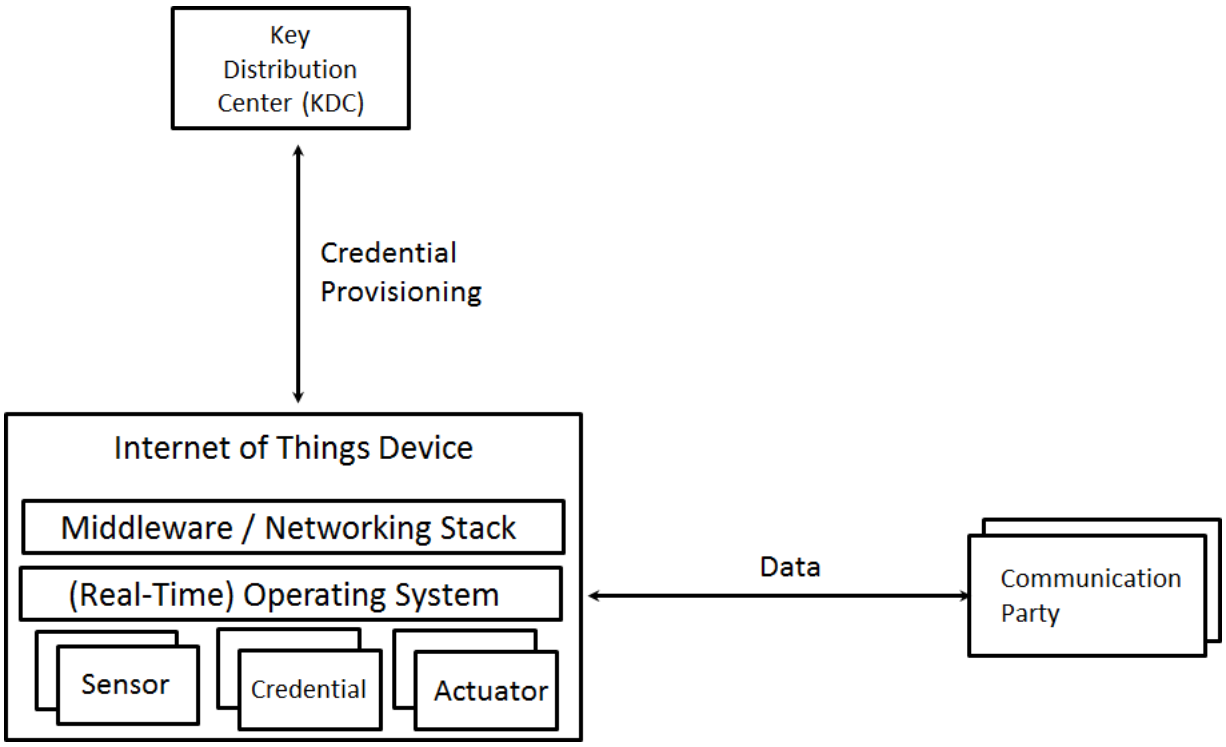

Figure 1: Architecture with a KDC.

Note: While credential management may happen through this central KDC, the actual data transmission still happens peer-to-peer.

The KDC itself is not necessarily a standalone device; it is a logical function that can be combined with other entities. For example, it can be bundled within a larger device management solution.

While the manufacturer credentials are long-lived or have no expiry date, the operational credentials are typically shorter-lived and for use with dedicated communicating parties. Unlike other computing devices, such as smart phones, IoT devices only have a small number of communication parties, which conveniently limits the number of operational credentials they have to store. It is common for IoT devices to connect to a server infrastructure offered by the manufacturer of the device and/or another cloud-based service. On-premise solutions for a KDC are also a viable deployment option in enterprise environments. However, other IoT communication patterns may require some functions of the KDC service to be run on a home gateway or, in some cases in the smartphone acting as a gateway.

Luckily, systems that standardize the KDC architecture have existed for a long time and have been tailored for use with IoT devices. The following sections explore different implementations of these systems, including:

- the Lightweight Machine-to-Machine (LwM2M) protocol [5] standardized by the Open Mobile Alliance (OMA)
- the Open Connectivity Foundation (OCF) [6],

- ACE-OAuth from the IETF ACE working group [7].

## Lightweight Machine-to-Machine (LwM2M)

Figure 2 shows the client-server architecture of LwM2M version 1.0 [5] with an IoT device acting as an LwM2M Client and the IoT service acting as the LwM2M Server. The LwM2M specification uses the Constrained Application Protocol (CoAP) [8] over UDP and optionally over SMS. DTLS [9] provides the state-of-the-art communication security infrastructure. The LwM2M Bootstrap Server plays the role of the KDC.

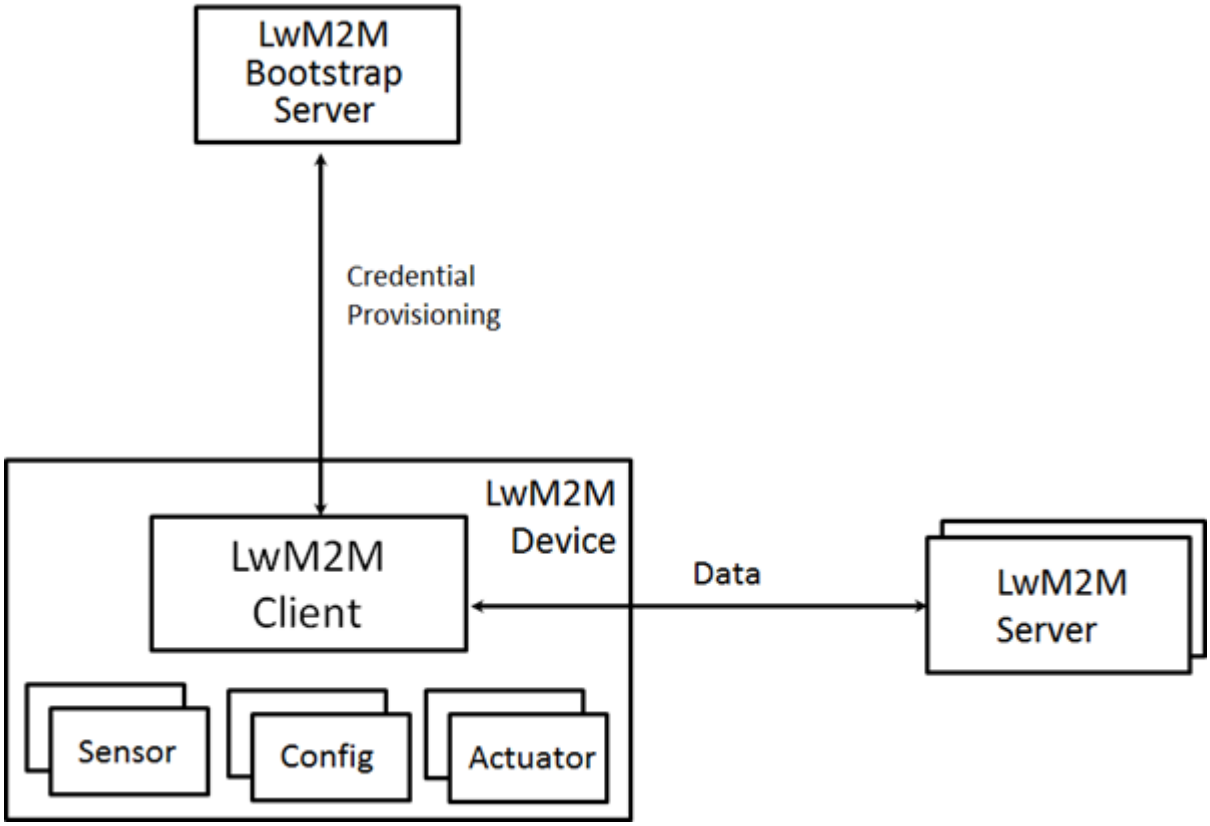


Figure 2: LwM2M v1.0 Architecture.

LwM2M supports a number of ways of provisioning the IoT device with operational credentials (the three types listed in Table 1). While the specification does not mandate how to store the credentials on the IoT device, it defines a data model that is a combination of a RESTful API and a description of objects and resources being manipulated. The RESTful API allows interaction between the LwM2M client and server. The LwM2M specification assumes that the credentials, either pre-shared secrets, raw public keys, or certificates, are pre-provisioned on the device during manufacturing, or by commissioning for use with a Bootstrap Server. The LwM2M Bootstrap Server authenticates the LwM2M client using the manufacturer credentials and provisions operational credentials to the device. Therefore, manufacturer credentials as well as operational credentials are maintained in the LwM2M Security object. The list of all defined

objects can be found in the core objects, which are contained in Appendix E of the technical specification [1]. While the LwM2M Security object is one of the most important objects in context of this whitepaper, there are some other important objects:

- The LwM2M Server object dictates which LwM2M and bootstrap server the IoT device is allowed to interact with.
- The Access Control object controls access to objects and resources (such as sensors and actuators) by LwM2M servers.

There are several questions that arise in the context of provisioning operational credentials to the device, namely:

- Who initiates the provisioning? The LwM2M Client on the IoT device (in the form of client-initiated bootstrapping) or the LwM2M Bootstrap Server (so-called server-initiated bootstrapping)
- What types of credentials are used with the bootstrap server? pre-shared secret, raw public keys or certificates
- Who creates the private keys? LwM2M Client on the IoT device or the LwM2M Bootstrap Server (In LwM2M version 1.0, the server always creates symmetric keys and provisions them to the client and hence we focus only on asymmetric keys in this question.)
- What types of credentials are created by the LwM2M Bootstrap Server for use with the LwM2M servers? pre-shared secrets, raw public keys or certificates

The pre-shared key, raw public key, and certificate credentials can be generated by the LwM2M Bootstrap Server and then provisioned to the LwM2M Client on the IoT device. For certificates, there is also an option to allow IoT devices to generate their own public / private key pair. The public key is then sent to the LwM2M Bootstrap Server as part of a Certificate Signing Request (CSR). The types of credentials cover a range of use cases, and the type of credential to use is determined by:

- the operational context,
- preferences,
- threat model,
- deployment environment.

The lifetime of the generated keys also varies, as shown in Table 2.

There is a qualitative difference between the two main credentials, namely the manufacturer credential and the operational credentials.

Table 2: Credential Lifecycle Management in LwM2M.

| | LwM2M Client ⇔ LwM2M Bootstrap Server | LwM2M Client ⇔ LwM2M Server |
|---|---|---|
| **Credential Type** | Manufacturer Credential | Operational credentials - asymmetric or symmetric keys |
| **Created By** | Device manufacturer | LwM2M Bootstrap Server or, in case of asymmetric keys jointly with the LwM2M device |
| **Lifetime** | Long-lived | Medium-lived. |
| **Refreshed By** | Device manufacturer/OEM (optional) | LwM2M Bootstrap Server |
| **Covered by LwM2M Specifications** | No (outside the scope) | Yes |

# Open Connectivity Foundation (OCF)

Figure 3 shows the client-server architecture of OCF version 1.0, with an IoT device acting as an OCF Client and another IoT device acting as the OCF Server. The OCF specification uses CoAP and DTLS. The Device Owner Transfer Service (DOXS) performs device onboarding / commissioning operations such as, configuring device settings for connecting to a production IoT network. Additionally, credentials and ACLs needed to securely connect to a Credential Management Service (CMS) and Access Management Service (AMS) are provisioned by the DOXS service. Both the DOXS and CMS play the role of the KDC. OCF devices can play the role of both Client and Server in the context of device management, allowing devices to proactively seek credentials, authorization and setup.

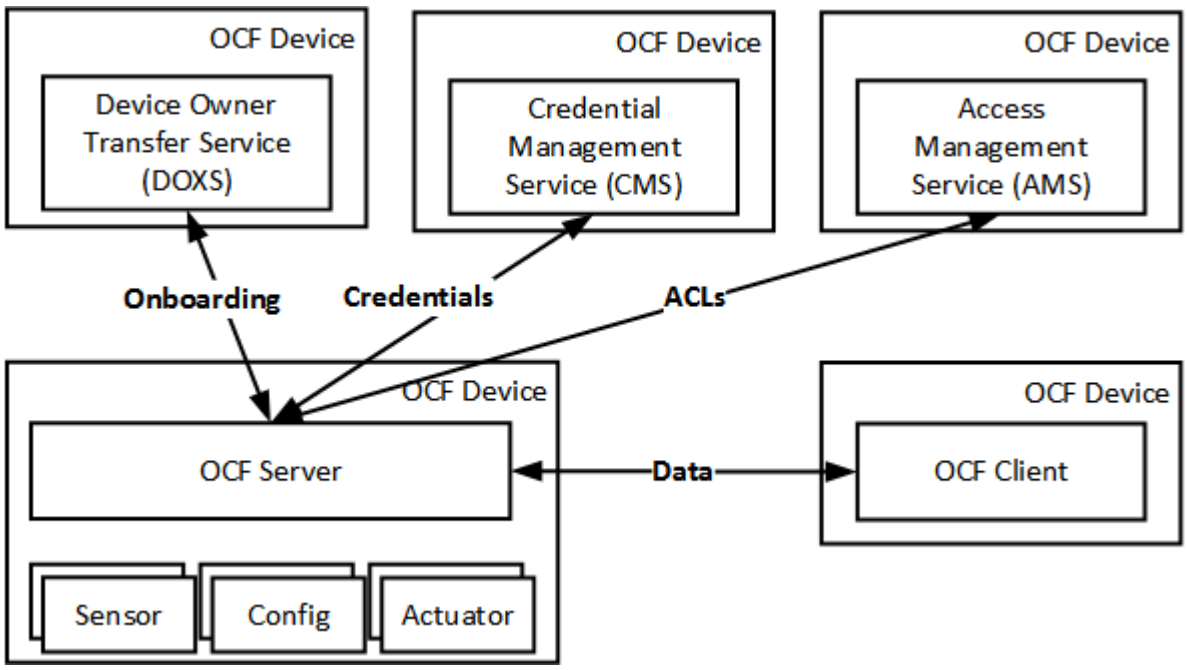

Figure 3: OCF v1.0 Architecture.

OCF supports a number of techniques to provision the IoT device with credentials and in addition supports the three credential types listed in Table 1. While the OCF specification does not mandate a specific way to store the credentials at the IoT device, in order to allow the provisioning of data to that device, it specifies a data model that supports a RESTful API. It also standardizes methods for onboarding / commissioning devices into an IoT network. Onboarding includes an exchange of values and credentials that establishes device ownership known as Owner Transfer Methods (OTM). OTMs may differ, but play an important role in defining the basis for trust in a device.

The OCF specification assumes that the long-term manufacturing credentials, either in form of pre-shared secrets, raw public keys, or certificates, are pre-provisioned to the device during manufacturing or by supply chain processes. The DOXS provider authenticates the OCF Device using manufacturing credentials and establishes owner credentials that are local to the IoT network into which the device is being commissioned. Manufacturing credentials, as well as owner credentials, are maintained in the OCF Security Resources. The list of all defined resources can be found in the OCF Security technical specification [9]. While the Security Resources are some of the most important resources in context of this whitepaper, there are some other important resources:

- The OCF resource discovery resource is used to discover devices that are not yet claimed by an IoT network.
- The device owner transfer resource is used to establish device ownership properties including which IoT network claims it, which owner transfer method was used to establish trust and which DOXS service to contact for device lifecycle support.

- The provisioning status resource is used to maintain device provisioning and operational status. OCF devices have well-defined operational modes where some aspects of the device capabilities are disabled while others are enabled. For example, a device that has not been onboarded will not, under normal conditions, be able to perform any other operation except onboarding. Operational modes contribute to the security and safety of the device.
- The credential resource is used to establish the set of credentials and trust anchors the device will use when establishing secure sessions with peer devices. It also identifies the Credential Management Service (CMS) that is authorized to maintain the credential resource.
- The access control resources are used to define finer grained access semantics to resources hosted by a device. It also identifies the Access Management Service (AMS) that is authorized to maintain the ACL resources.

There are several issues in provisioning credentials to OCF devices:

- Who initiates the provisioning? An OCF Service (the authorized OCF Client) that connects to an OCF Server device (in the form of client-initiated provisioning) or the OCF Server that connects to an authorized OCF Client (server-initiated provisioning).
- What types of credentials are supported and for what tasks? Table 3 identifies three classes of credential used in OCF, namely manufacturer, owner, and operational credentials
    - OTMs define the type of credentials needed to establish trust in the device. There are a number of techniques available to device manufacturers to facilitate the transfer of the device through a supply chain and to attest to device integrity and hardening properties. An OTM must construct a secure channel over which the remainder of the ownership transfer and onboarding exchange occurs.
    - Owner credentials are local to the IoT network that claims the new device. Owner credentials may be symmetric or asymmetric, according to the policy of the network operator. The OTM session provides a secure context within which the owner credentials are provisioned or derived.
    - Operational credentials may be symmetric or asymmetric, according to the policy of the CMS (aka network operator). The CMS provisions credentials directly over a DTLS session using the CMS credential provisioned during onboarding or indirectly using certificates issued locally by the CMS acting as a Certificate Authority (CA) or an existing local CA service.
    - Session keys are generated as part of a DTLS handshake involving Operational credentials. Session keys protect data between the session endpoints. The OCF framework is required to be the session endpoint.
- Who creates the private keys? OCF devices generate asymmetric private keys and enroll the corresponding public key with the CMS (acting as CA).
- What types of credentials are created by the CMS for use with the OCF devices? Pre-shared secrets, raw public keys or certificates.

Table 3: Credential Lifecycle Management in OCF.

| | OCF Device ⇔ DOXS & Credential Management Service | | OCF Client ⇔ OCF Server |
|---|---|---|---|
| **Credential Type** | Manufacturer Credential (for use between the DOXS Service to onboard / commission the device). | Owner Credential - asymmetric or symmetric keys established between OCF Device and DOXS Server. | Operational credentials - asymmetric or symmetric keys established between OCF Client and OCF Server. |
| **Created By** | Device manufacturer. | OCF Device & DOXS Server during Owner Transfer Method (OTM). | CMS (CMS credential is created by DOXS). |
| **Lifetime** | Long-lived. | Long-lived. | Medium-lived. |
| **Refreshed By** | Device manufacturer (optional) | DOXS Server | CMS (CMS credential is refreshed by DOXS) |
| **Covered by OCF Specifications** | No (outside the scope) | Yes | Yes |

There is a qualitative difference among the three credential types in OCF where the longest lived keys (which are potentially slowest computationally) are used less frequently than the shortest lived keys (which are potentially fastest computationally). The OCF credential hierarchy consists primarily of the manufacturer credential, which attests to trust properties of the device at the time of onboarding / commissioning into the IoT network, but have no significant role thereafter unless the device is re-onboarded / re-commissioned. A primary function of onboarding is execution of an Owner Transfer Method (OTM) that performs a key exchange protocol whereby the owner credentials are established. Owner credentials facilitate secure provisioning (and re-provisioning) of the local device and security management service providers.

In summary, OCF service providers, clients and servers are OCF devices. All secure interactions between OCF devices rely on operational credentials. Operational credentials are used in conjunction with DTLS.

# ACE-OAuth

Complementing the work on LwM2M and OCF, the IETF ACE working group tackles use cases that require user identity management, consent of the user, and user authentication. This solution is referred to as ACE-OAuth since it represents a profile of the widely deployed OAuth technology. The classical scenario is a physical access control solution in an enterprise where existing enterprise identity management infrastructure is re-used for controlling access to facilities, like office rooms and labs. To involve an end user in the interaction often requires their smartphone, wearable, or tablet, and it starts with user authentication where the user logs into their regular identity management system (unless already logged in for other purposes) and then solicits access to whatever protected resources, such as access to a meeting room. The owner of the resource, such as the facility manager or IT manager, needs to provide consent (unless done in advance). In this case, the resource requestor and the party granting consent are often two different entities.

Once consent has been granted, the app on the smart phone obtains an access token (and a refresh token) for use with the access control system, such as for a door lock. Note, however, that this exchange also involves the ACE-OAuth client, which needs to authenticate itself to the authorization server using client secrets (in case of a confidential client at least). These client secrets need to be pre-established between the ACE-OAuth client and the authorization server. This access token is then used in future interactions unless revoked or expired. A refresh token is used to request further access tokens once they are expired without involving humans again in the process.

To evaluate requests to protected resources requires authorization policies to be present at the authorization server. These policies can be created on the fly, as part of the consent process, or can be pre-populated. Which approach is suitable for a given scenario often depends on the preferences of those deploying these services.

This solution aligns better with approaches used today on smartphones by native apps, allows cross domain identity management, independent user authentication mechanisms, and fine-grained access control techniques.

The architecture of ACE-OAuth, shown in Figure 4, intentionally aligns with the OAuth framework. To support constrained IoT devices, the ACE working group introduces the use of CoAP (either instead of, or in addition to the use of HTTP, as used in classical OAuth) and Concise Binary Object Representation (CBOR) encoding of access tokens. This architecture also uses proof-of-possession (PoP) tokens that are tied to a key, instead of bearer tokens which are used on the web and on smart phones. PoP tokens provide better security against token leakage than bearer token, when keys are properly protected.

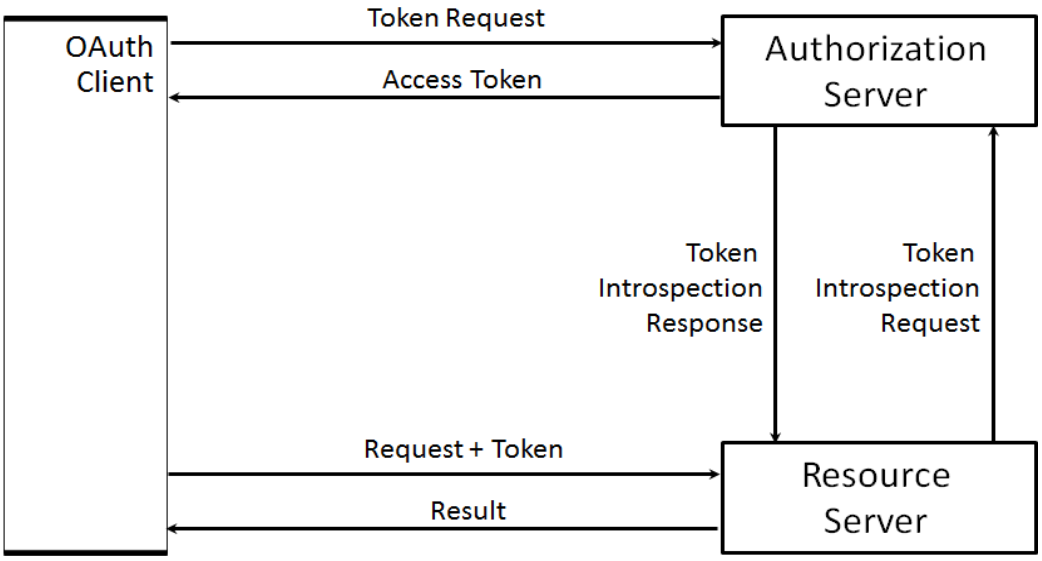


Figure 4: ACE-OAuth Architecture.

An example instantiation of the architecture shown in Figure 4 is to run an OAuth Client on a smartphone bundled with another app and to interface with the resource server (such as a door lock) via Bluetooth Low Energy. The authorization server in OAuth issues access tokens in accordance with the authorization policies and the consent of the user. Due to the nature of PoP tokens, the authorization server also becomes a KDC.

More details about the ACE-OAuth architecture can be found in [7]. A standardized format for the access token is available with the CBOR Web Token (CWT) [10]. Information about implementing OAuth on native apps securely can be found in [11] and the CWT PoP token functionality is available with [12].

ACE-OAuth augments a device management solution like LwM2M since the IoT device, which plays the role of a resource server, still needs to be managed. It is important to note that the ACE-OAuth architecture assumes there is a credential provisioned at the participating parties. In this architecture, the:

- resource server needs to verify access tokens created by the authorization server and needs to be in possession of credentials,
- client needs to be authenticated by the authorization server,
- authorization server needs to authenticate itself to the client and needs to possess keying material to protect the access token.

Credentials for use by the IoT devices may be provisioned with a standard such as LwM2M.

There are several issues for provisioning credentials to ACE-OAuth devices:

- Who initiates the provisioning? The user and, in terms of the protocol entities, the client.

- What types of credentials are used with the Authorization Server? pre-shared secret, raw public keys or certificates.
- Who creates the private keys? The client creates private keys related to PoP tokens and gets the Authorization Server to sign the corresponding public keys by having them included into the PoP token.
- What types of credentials are created by the Authorization for use with the Resource Servers? pre-shared secrets, raw public keys or certificates.

Since ACE-OAuth also uses credentials itself, Table 4 provides further details about their purpose and properties.

Table 4: Credential Lifecycle Management in ACE-OAuth.

| | **Authorization Server ⇔ Resource Server** | **Client ⇔ Authorization Server** | | **User ⇔ Authorization Server** |
|---|---|---|---|---|
| **Credential Name** | Access Token (or PoP token) | Client Secret (mandatory for confidential clients) | Refresh Token (optional) | |
| **Credential Type** | Operational Credential representing delegated access rights (similar to owner credential) | Manufacturer Credential | Operational Credential | User Credential (such as username and password or more secure technology like FIDO) |
| **Created By** | Authorization Server | Manufacturer | Authorization Server | User Identity Management System |
| **Lifetime** | Short lived | Long lived | Medium lived | Long lived |
| **Refreshed By** | Authorization Server | Manufacturer (optional) | Authorization Server | User Identity Management System |
| **Covered by ACE-OAuth Specifications** | Yes | No | Yes | No |

# The Role of DTLS in Key Management

Manufacturer and operational credentials are typically not used directly to protect application data, at least not in the protocols presented in this whitepaper. Instead, they are used as input to a security protocol, like DTLS[i]. DTLS itself, like other mature security protocols, adds another layer of credentials for performance reasons. Since all three architectures presented in this

whitepaper make use of DTLS, it is worthwhile to highlight its role in the context of key management.

The long term credentials, which authenticate the communicating parties, are used as input to the DTLS handshake. In IoT scenarios, this will typically be mutual authentication, so that the client authenticates to the server and vice versa. These credentials are used during the full handshake. Subsequently, credentials are created for use with session resumption. As these session resumption credentials are based on symmetric cryptography, they avoids the cryptographic overhead of a full handshake.

Finally, with the completion of a successful DTLS handshake credentials are established for confidentiality and integrity protection of application data. They , are also based on symmetric cryptography.

# Conclusion

The key points in this whitepaper are:

- The ability to dynamically provision operational credentials to IoT devices is important for manageability, scale, security and convenience. Offering automatic key management without involving humans is an important step in improving security of IoT devices. One important commonality in all three presented solutions (LwM2M, OCF and ACE-OAuth) is that they make use of manufacturer credentials to provision operational credentials on IoT devices.
- Since the KDC architecture is a well understood concept, it is now applied to the IoT environment in a way that suits the constraints of these devices and networks. Three instantiations of this architecture have been presented with LwM2M, OCF and ACE-OAuth.
- Authorization management and key management are closely related. LwM2M, OCF, and ACE-OAuth have RESTful models that allow for flexible key management using Bootstrap Server, Credential Management Service (CMS) and an Authorization Server, respectfully. OAuth-based authorization exchanges can be implemented over the top of IoT frameworks leveraging their object model(s) to define token contents and semantics. Nevertheless, the OAuth message orchestration offers a context for message exchanges that aligns with several accepted models for dynamic authorization management.

The presented solutions use a number of credentials, in a pseudo hierarchical manner, that offer trade-offs between computationally intensive functions (such trust establishment and authentication across an ecosystem), and less resource intensive functions (such as localized authentications and short term data protection). The authors believe a key management architecture that incorporates such trade-offs is necessary to effectively optimize for constrained devices. While there may be significant performance costs paid up front, at device onboarding / commissioning, the amount of time spent onboarding / commissioning is relatively small

compared to the time spent on short term symmetric key operations performed over the lifetime of the device.

All the presented solutions offer a distinction between manufacturer credentials, for establishing attestable trust in the device (in terms of hardware, software and manufacturer) and operational credentials, for device-to-device authentication and negotiation of session keys. Symmetric session keys are used for protection of application data using protocols like DTLS to offer confidentiality, integrity and data-origin authentication.

# References

[1] ISOC, "The Internet of Things (IoT): An Overview", 15 Oct 2015, URL: https://www.internetsociety.org/resources/doc/2015/iot-overview
[2] H. Tschofenig, et al., "Architectural Considerations in Smart Object Networking", RFC 7452, March 2015.
[3] R. Housley, et al., "Trust Anchor Management Protocol (TAMP)", RFC 5934, August 2010. URL: https://tools.ietf.org/html/rfc5934
[4] E. Ronen, et al., "IoT goes nuclear: creating a ZigBee chain reaction", ePrint Report 1047, 2016.
[5] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification, Approved Version 1.0, OMA-TS-LightweightM2M-V1_0-20170208-A", 8. Feb. 2017, URL: http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf
[6] OCF, "OCF SPECIFICATION 1.0", June 28, 2017, URL: https://openconnectivity.org/resources/specifications
[7] L. Seitz, et al. "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-07 (work in progress), August 2017. URL: https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-07
[8] Z. Shelby, et al, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014, URL: https://tools.ietf.org/html/rfc7252
[9] E. Rescorla, N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012, URL: https://tools.ietf.org/html/rfc6347
[10] M. Jones, et al, "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-08 (work in progress), August 2017. URL: https://tools.ietf.org/html/draft-ietf-ace-cbor-web-token-08
[11] W. Denniss, et al., "OAuth 2.0 for Native Apps", RFC 8252, October 2017.
[12] M. Jones, et al., "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-00 (work in progress), July 2017, URL: https://tools.ietf.org/html/draft-ietf-ace-cwt-proof-of-possession-00

[i] The currently described solutions make use of DTLS but they are expected to also support TLS in the future. DTLS is the preferred approach for offering communication security of datagram protocols.