An Application-Layer Approach to End-to-End Security for the Internet of Things

Approved – 24 Oct 2019

Open Mobile Alliance
OMA-WP-e2e_Sec_IoT-20191024-A

Contents

1. Introduction

Any business relying on the Internet of Things (IoT) for their operations requires that solutions are reliable and trustworthy. Security is a prerequisite and protection of online industrial or enterprise assets cannot be overstated. This paper focuses on the need to secure application data end-to-end between IoT device and enterprise application or IoT service provider. Securing application data applies to messages exchanged between applications of two endpoints, specifically **sender authentication**, **receiver authentication**, **message integrity** and **message confidentiality**. In this paper, *end-to-end authentication* means that a receiver can determine the sender of a message - and vice versa. In IoT, a networked application service authenticates the application-layer sender of a message. *End-to-end confidentiality* means that only the receiving endpoint can read the sender's message. In IoT, an endpoint itself isn't always reachable except through a gateway, which serves as a *proxy* to the endpoint device. A gateway may translate addresses, protocols, or commands along the IoT service path. As various types of gateways proliferate in IoT services, the communication security between sender and receiver depends on the security of each hop combined with business agreements and trust relations between the involved parties.

Such an aggregation of hop-by-hop security chains allows some aspects of authentication, message integrity and confidentiality sufficient for many use cases, but insufficient for others. IoT proxies such as **application-layer gateways** and **middleboxes** may alter the messages they forward at various protocol layers in the stack. Security may ultimately degenerate to the security of a set of gateway hops. Hop-by-hop security offers more points of attack and is a greater risk to the IoT service, therefore some applications do require even stronger security.

Secure messaging is critical for safety, comfort, and control systems. Secure messaging includes privacy: The EU General Data Privacy Regulation [GDPR] mandates the protection of personal data for their data subjects. The theme of this paper is that IoT data must be secure in transit and securing data at the transport layer alone is not always sufficient for preserving integrity and confidentiality through proxies, gateways or other middleboxes. Application-layer[1] security is needed for many IoT service topologies to prevent critical data from becoming unprotected in middleboxes.

The next section, Section 2, takes a closer look at what needs to be protected and in which environments. Section 3 gives real-world use cases for application-layer end-to-end IoT security. Section 4 describes the Open Mobile Alliance solution to these types of use cases, which is Lightweight M2M (**LwM2M**) application-layer security based on OSCORE. Lightweight Machine-to-Machine (LwM2M) is a framework for M2M and IoT device management and service enablement [LwM2M_Core][LwM2M_Trans]. Section 5 considers key management for OSCORE. The Conclusion follows and shows how LwM2M application-layer security applies to the presented use cases.

[1] Protocol layers are in this whitepaper referring to the Internet protocol stack

2. End-to-End Security for IoT

Some Section 3 use cases need secure messaging at multiple, overlapping communications layers. **Link layer security**, such as Wi-Fi WPA2, ensures that only authorized parties can use a particular network. A wireless network is usually one of multiple networks between IoT endpoints. **Transport-layer Security** (TLS)[2] works across those networks and provides end-to-end security in many use cases, but each middlebox may become an endpoint that may terminate a TLS connection. In this way, middleboxes destroy transport-layer end-to-end authentication, integrity and confidentiality by creating more than one transport connection end-to-end. In this case, an IoT service needs more than TLS to achieve end-to-end security: **Security at the application layer** preserves end-to-end security over middleboxes and IoT gateways.

Security is applied to the application layer to make it unchangeable and unreadable between application endpoints. When middleboxes gain unrestricted access to the entire message without needing all the data in the message, this violates the *principle of least privilege*. Application-layer forwarding does not need disclosure of application layer data contents at all.

IoT services often include **constrained devices**, which are constrained in power, computation and networking capabilities [RFC7228]. For example, security at three layers may be too demanding of these resources. What layers can be skipped depends on the needs of the particular application and environment. The following subsections consider what applications need to protect, where they need to protect it, and when.

2.1 What Needs to be Protected?

Different types of IoT assets need to be protected. IoT devices and user interfaces are **IoT-endpoint assets** such as light bulbs, mobile phones, doorbells, locks, commercial and industrial sensors and actuators. IoT gateways, brokers, proxies, and firewalls are **middlebox assets**. Endpoints and middleboxes create, consume or process **information assets**, such as application-layer payload, metadata, and addressing information. This paper focuses solely on information assets and assumes proper operational security in endpoint and middlebox assets.

Endpoints that depend on the security of middleboxes share fate with them: Middleboxes are common targets of mass-surveillance [Constantin], extortion [Mirai], and other attacks. Thus, each middlebox hop of decryption, processing, and re-encryption incrementally increases IoT security risks. There's more risk when middleboxes can read more message data than they need or endpoints collect more data than they need. *Privacy-by-design* minimizes collected data according to a documented need. Data are risky and even *toxic* [Schneier] to organizations. This paper assumes secure data design, but data design is outside the paper's scope, which is protecting IoT messages that contain the data regardless of data architecture.

In this paper, protection is application-layer message authentication and confidentiality. It also includes protection against message replays. Data about the message also need some protection: Metadata such as instruction/error code, content format or other application related information expressed in the application-layer header fields typically need at least integrity protection. Source and destination network addresses, host addresses, and port numbers typically exist outside of the scope of the application and are often used by middleboxes along the service path such that they should neither be encrypted nor integrity protected: Message routers, for example, filter on address and other fields; forwarding proxies may change destination addresses.

Some metadata, moreover, reveal personal information or other confidential data that only intended endpoints should read. Other metadata are mutable and don't figure into endpoint authentication of the message. Hence, not all metadata must be protected end-to-end, but LwM2M application security and comparable protocols must properly protect metadata that do.

Write operations to devices, such as device configuration or actuation, may require additional security considerations. For example, a physical lock does not only need to be protected against replay, it must also be able to verify that an unlock request is fresh to avoid delayed unlock requests from being accepted. Furthermore, to verify the current state of a device requires a binding of response to request, so that delayed responses are not accepted.

2.2 When and Where Does It Need to be Protected?

Some IoT applications resemble email and chat services: Like a PC, an IoT device may be intermittently offline to save power, or due to a low-powered or lossy network. In many cases, transport-layer security doesn't protect store-and-forward message delivery end to end, just hop-by-hop. Many IoT messages are delivered over a connection with data sent and

[2] Refer to [TLS] and [DTLS] for details on transport-layer security.

received in real time. Thus, both **connection-oriented** and **store-and-forward** IoT messages need to be protected. IoT data need protection in-transit and at-rest.

TLS is well suited for messages over single-hop connections but is not always sufficient for multi-hop connections. Application layer security protects store-and-forward messages and messages over multi-hop connections.

IoT data in transit need to be protected between endpoints end-to-end. IoT data at rest need to be protected within an endpoint. As stated above, this paper is concerned with IoT data in transit that are acquired from consumer and enterprise devices. These devices run on a variety of networks, some low-powered and lossy networks.

3. Use Cases

In this section we present three use cases where there is a need to protect messages end-to-end between sender and receiver. These use cases involve different types of middleboxes translating between different types of transport and application layer protocols, performing firewalling, and handling address translations. In all use cases the LwM2M framework is used. LwM2M uses the Constrained Application Protocol (CoAP), a RESTful constrained application-layer protocol much more compact than HTTP due to its binary encoding [CoAP], as application transfer protocol.

3.1 Explosive Gas Detection

A city operator wishes to reduce incidents related to buildup and ignition of explosive gases in vaults such as building basements or below grade conduit vaults. The economic return is only generated by deploying low cost monitoring devices across a wide area, necessitating constrained and embedded devices underneath an available wide-area network canopy that can reliably reach these difficult locations, such as a Wi-SUN canopy.

The system architecture for the explosive gas detection is shown in Figure 1. The network technology is gated via an application layer proxy at the head-end, hereby referred to as a "Head-end CoAP Proxy," which understands how to manage and coordinate traffic to the network. This includes routing, traffic rate control, congestion control, device authorization (e.g. who may transmit to which devices), and device-availability detection. Using CoAP over TLS over TCP from the Head-end CoAP Proxy to the LwM2M Server may be necessary to traverse firewalls, as explained in [CoAP-TCP].

A second proxy is located at the device itself. A single physical device package in fact represents a small constellation of actual sensors. Each sensor won't have the capacity to fully manage a LwM2M endpoint including a full networking stack, so a common network interface that links these sensors serves to proxy network traffic. The individual sensors handles CoAP over serial, thus the network interface is a CoAP proxy. Each individual sensor contains a LwM2M client implementation enabling communication with the LwM2M server. This may be a LwM2M client with only the minimal functionality for reporting data. In this design, the network interface itself is also a sensor with its own LwM2M client.

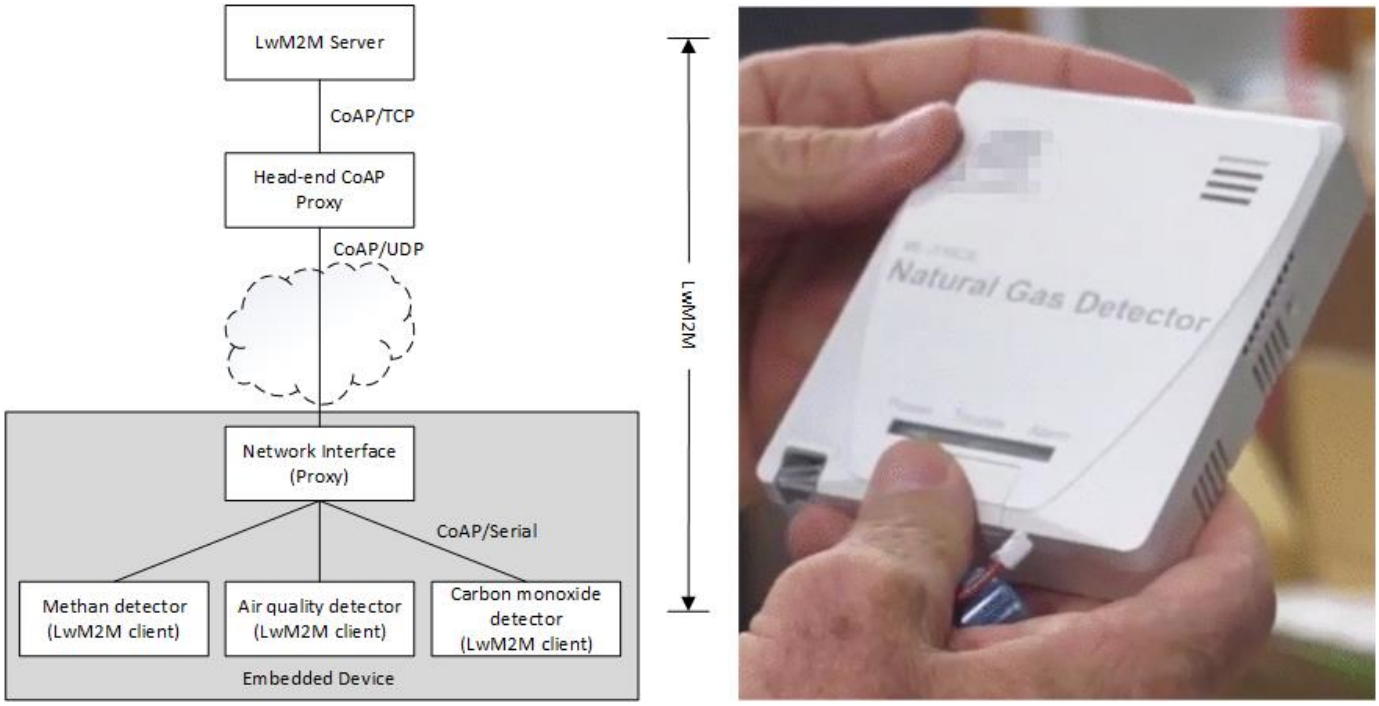

Figure 1 System architecture for the explosive gas detection use case and picture of a gas detector embedded device

This specific use case utilizes two proxies, where the head-end has a network access proxy and the mobile-end has a device access proxy. Double proxies obscure the source and destination endpoints of a message, which a proxy typically handles in plain-text form. The ability to read and change address data enables the above decribed application-layer proxy services.

The exposure of message contents in these proxy middleboxes has all the risks and threats described in Section 2, above. The benefit of the end-to-end security approach is that there are multiple entities involved who do not necessarily trust each other. The LwM2M server may have a relationship with the sensor provider, and both are depending on a network provider for service, but either or both may not want the network provider to have access to the sensor data, or being able to modify it.

3.2 Healthcare Monitoring

In a remote healthcare monitoring system, physiological parameters of patients such as heart rate, blood pressure, respiratory rate, and body temperature are constantly monitored. Constrained healthcare monitoring devices connect to report sensor data using NB-IoT [NarrowBandIoT], which is a mobile radio access technology targeted for massive deployment of IoT devices. The modem subsystem of each constrained device is mostly sleeping and periodically wakes up to report one or a few bytes of sensor data collected by the low-power sensor subsystem. This happens at most a few times per hour. With such small payloads, the overhead of the TCP/IP stack becomes significant and the "Non-IP Data Delivery (NIDD)" transport mode of NB-IoT eliminates TCP/IP packet headers from data being sent by the device. LwM2M is used for sensor data reporting and Non-IP Data Delivery mode is supported in LwM2Mv1.1.[3] When NIDD is used, LwM2M/CoAP messages are carried end-to-end between the LwM2M server collecting sensor data via the cellular network to the monitoring device that supports CoAP and contains a LwM2M client. An example system architecture is shown in Figure 2.

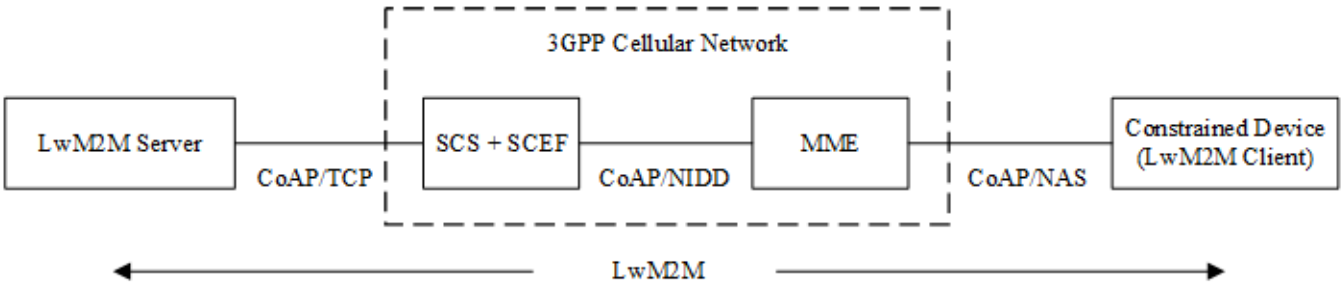

Figure 2 System architecture for the healthcare monitoring use case

The Service Capability Exposure Function (SCEF) exposes the services and capabilities of the 3GPP cellular network. The NIDD service is used for communicating with the health monitoring devices where the devices send and receive CoAP messages without use of IP. These messages are conveyed as signaling data in the Non-Access Stratum (NAS) signaling protocol. NAS signaling occurs between the devices and the Mobile Management Entity (MME). The CoAP messages are then relayed further to SCEF. Co-located with SCEF is the Services Capability Server (SCS) that provides additional services for use by LwM2M servers and other servers. SCS is the interface towards the LwM2M server and uses SCEF services including the NIDD service for sending and receiving data from the healthcare monitoring devices. SCS acts as a proxy allowing the devices to communicate with other LwM2M servers in addition to the LwM2M server for healthcare sensor data reporting. An example of such a server is a LwM2M server handling device management. The SCS investigates CoAP messages from the devices to determine the destination of each CoAP message, assigns an IP address for each device, adds TCP/IP headers to the CoAP messages, and forwards the messages to the targeted LwM2M servers. The SCS also provides a way for LwM2M servers to send CoAP messages to constrained devices, monitor devices, and retrieve notifications when devices are connected.

3.3 Protected Communication with the Web

Healthcare information is typically sensitive from a privacy point of view and the provider of the healthcare monitoring service wants to make sure that sensor data is not accessible and modifiable by the cellular network operator providing the NIDD NB-IoT service. That operator's SCS, SCEF and MME are examples of middleboxes as Section 1 defines them.
Protected Communication with the Web

Many IoT deployments need to communicate with the Web, because they have to upload data to the Cloud or because they are controlled by a Web application.

Communication between a web application talking HTTP and an IoT appliance speaking CoAP is facilitated by a cross-protocol proxy or gateway translating between the protocols. However, translation means any TLS connection must be

[3] For more details on the use of 3GPP cellular NIDD in LwM2M, refer to [LwM2M_CIoT] and [LwM2M_Trans].

terminated at the translation proxy. If the translation proxy has access to the plaintext HTTP/CoAP message, then end-to-end security between device and cloud is broken, and the risks and threats described in Section 2 applies.

The transport of LwM2M over HTTP is not specified in general, but CoAP messages can be mapped to HTTP. It is therefore possible to transport LwM2M operations over HTTP. The procedure is based on the following invertible mappings:

A. A LwM2M operation is mapped to a CoAP message as specified in the LwM2M Transport Specification [LwM2M_Trans]
B. A restricted set of CoAP messages can be mapped into HTTP messages, as described in the CoAP specification [CoAP] and the HTTP-CoAP mapping guidelines [HTTP-CoAP].

By combining A and B above, a LwM2M operation can be mapped to CoAP and transported over HTTP. The HTTP message may pass one or more intermediaries, e.g. firewalls, until it reaches an HTTP-CoAP proxy, after which the LwM2M operation is transported over CoAP. The CoAP message may pass one of more intermediaries until it reaches the CoAP endpoint, where it is readily mapped to the LwM2M operation. The inverse mapping applies to messages going in the reverse direction. An example system set-up illustrating the above transport of LwM2M messages is shown in Figure 3. As will be shown in later sections, by using the above mapping of messages it is possible to protect messages end-to-end between two LwM2M endpoints. Note that additional TLS may be used with the HTTP hops if required by policy.

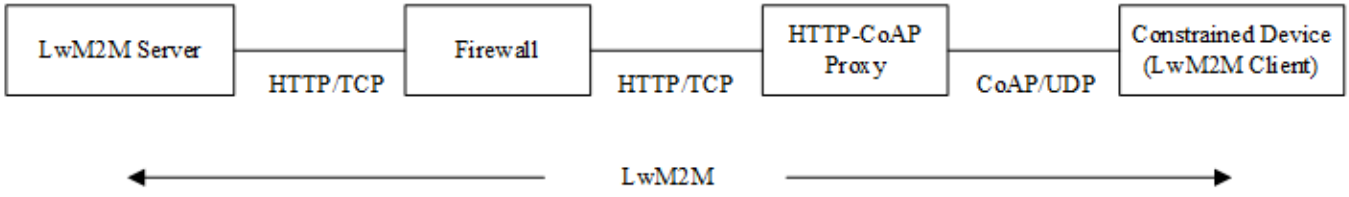

Figure 3 Example system setup for protected web communication use case where LwM2M is end-to-end

In general, one cannot map completely between HTTP and CoAP even though both are RESTful. The mapping, as currently defined, is restricted to a subset of HTTP and CoAP. Consequently, this procedure applies only to the LwM2M operations which are mapped to this subset based upon a common set of semantics. But the only LwM2M operations that are not mappable are those of the Information Reporting interface. This interface feature is based on the CoAP Observe option for which the mapping to HTTP is not defined.

In the above scenario, the application is LwM2M aware and we have LwM2M end-to-end and handle translation between HTTP and CoAP. In situations when the LwM2M architecture is integrated with other IoT systems the application is typically not LwM2M aware and there may be the need to translate between some other protocol and LwM2M. Examples of such protocols are MQTT, proprietary REST based protocols over HTTP, and SOAP over HTTP and the mapping of messages between the protocol and LwM2M affects the complexity of an end-to-end security solution.

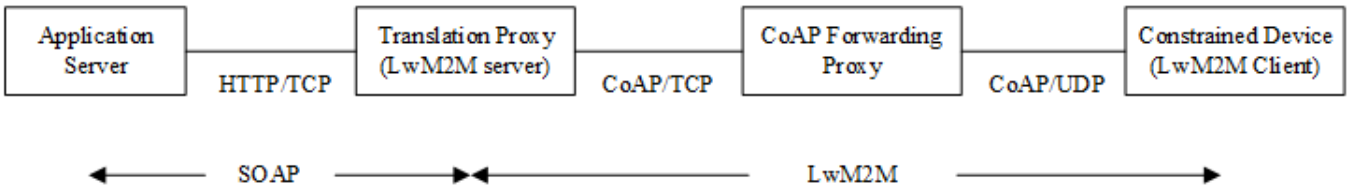

Figure 4 Example system setup for protected web communication use case where LwM2M is not end-to-end

An example of a hybrid architecture that uses SOAP, a non-RESTful protocol, and RESTful LwM2M is shown in Figure 4. Here a translation proxy translates from SOAP over HTTP into LwM2M over CoAP. (Note that HTTP to CoAP translation does not need to coincide with SOAP to LwM2M translation as in Figure 4).

4. End-to-End Application Layer Security in LwM2M

Lightweight Machine-to-Machine (LwM2M) is a framework for M2M and IoT device management and service enablement [LwM2M_Core][LwM2M_Trans]. It defines the application layer communication protocol between a LwM2M server and a LwM2M client, which is located in a LwM2M device. The LwM2M *enabler* enables device management and service enablement protocol exchanges between server and client. The enabler mainly targets resource constrained devices with a lightweight and compact protocol and an efficient resource data model. LwM2M uses CoAP as the application transfer protocol. To secure communication between a LwM2M client and a LwM2M server, LwM2M v1.0 defines transport-layer security using TLS/DTLS. The LwM2M framework includes a LwM2M bootstrap server that provisions key material used to protect the communication between LwM2M clients and LwM2M servers. Application-layer end-to-end security may be added on top of the existing LwM2M stack, in which case it would use a mechanism specifically designed and/or standardized for a particular use case or industry. LwM2M v1.1 extends LwM2M with application-layer security using Object Security for Constrained RESTful Environments (OSCORE) [OSCORE] which offers a unified mechanism usable across use cases and industries. OSCORE is a security protocol that protects CoAP message exchanges. OSCORE provides end-to-end security even with proxies in the service path between two LwM2M endpoints and even with different transports in the path. OSCORE is applicable to protocol messages that can be mapped to CoAP or a subset of CoAP, including LwM2M and HTTP. OSCORE also works between LwM2M endpoints and non-LwM2M endpoints in which case the intermediate node maps messages from a non-LwM2M end-point device into LwM2M operations and responses.

4.1 CBOR and COSE

OSCORE builds on Concise Binary Object Representation (CBOR) [CBOR] and CBOR Object Signing and Encryption (COSE) [COSE]. CBOR is a data format for the representation of binary data that's well-suited to constrained IoT devices. It was designed to make the code size for the encoder and decoder small and also to obtain a small message size. Still, the representation allows encoding of the most common data formats used in Internet standards such as basic types like numbers and strings. The CBOR data model is built on the JSON data model allowing easy translation between the two formats.

CBOR Object Signing and Encryption (COSE) describes how to protect messages with encryption, signatures, and message authentication code (MAC) operations. COSE utilizes the CBOR data format and also describes how to encode keys. COSE is functionally equivalent to the JSON Object Signing Encryption standard (JOSE). But unlike JOSE, COSE is tailored for constrained devices and uses binary encodings for binary data rather than Base64-based encoding as in JOSE; binary encoding minimizes the message size. COSE also uses a single “top layer” data format structure that is repeated as needed in sub layers enabling a small implementation footprint.

The COSE object structure consists of three parts. The first two parts are the set of (integrity) *protected header elements* and the set of *unprotected header elements*, which are not integrity protected. The header elements provide information about the message to the recipient that is not considered part of the payload of the message; this information is message metadata. Such metadata provide information about content type and about key, IV, and algorithm used in the protection.

The third part of a COSE encoding is the *plaintext* or *ciphertext message content*. Alternatively, the actual content can be detached and not contained in the message. The structure of the message content depends on the type of message. COSE defines six different types of messages for signed content, encrypted content, and MAC-protected content. Examples of the COSE object format for different message types are shown in Figure 5.

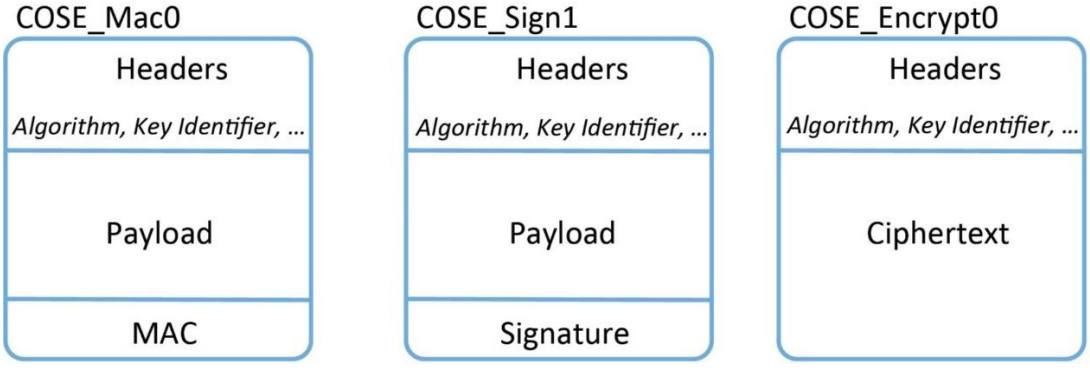

Figure 5 COSE object format for different message types

A COSE object structure uses layers to separate different types of cryptographic elements such as the content layer, which carries the encrypted plaintext, and the recipient layer, which contains the encrypted content encryption key (CEK). The same techniques and structures serve to encrypt both the application content and the CEK which minimizes the encoder and decoder code size. The COSE_Encrypt0 format in Figure 5 is a single layer structure where the recipient layer is left out and may be used only when the CEK is pre-shared.

4.2 OSCORE

A classical approach to protect application data and metadata is to bundle the application data with the metadata, encrypt and integrity protect these data, and send the output as the payload of the application transfer protocol. Replicate in plaintext, e.g. by including as part of the header or other non-protected parts of the application transfer protocol, those metadata that are needed by the middleboxes. This approach enables protocol translation at middleboxes but also add to the number of bits sent which is an important factor for constrained IoT devices where the life-time of battery powered devices may be more than 10 years without battery change. When protecting CoAP messages on application layer we want to avoid replication of metadata and also protect against attacks such as replay and out-of-order message injection. In CoAP there is sensitive metadata that needs protection end-to-end carried outside of the CoAP payload. The CoAP URI, for example, when used with LwM2M indicates to the application the target LwM2M object and resource, which it owns, and sensitive data like the REST method indicating the operation to be performed on the resource. Such data (at least parts of it) needs to be both encrypted and integrity protected end-to-end.

COSE has been designed to work well for protecting CoAP messages to constrained devices. COSE may protect only the CoAP payload, but it does not protect metadata of the CoAP protocol. The IETF recently finalized the standardization of "Object Security for Constrained RESTful Environments (OSCORE)" specifying how to use COSE to protect CoAP messages, including metadata in CoAP header and options [OSCORE]. The OSCORE specification describes the formats of the protected CoAP messages, which CoAP header fields should be protected or not, the CoAP options that should be encrypted and integrity protected, and the CoAP options that must be left unprotected to allow proxy modification. OSCORE protects the (logical) CoAP Request/-Response layer only, and not the (logical) CoAP Messaging Layer (see [CoAP] for logical layer description). The binding to the transport-layer protocol is left unprotected which means that OSCORE can be applied to CoAP even though the transport changes in different hops between sender and receiver.

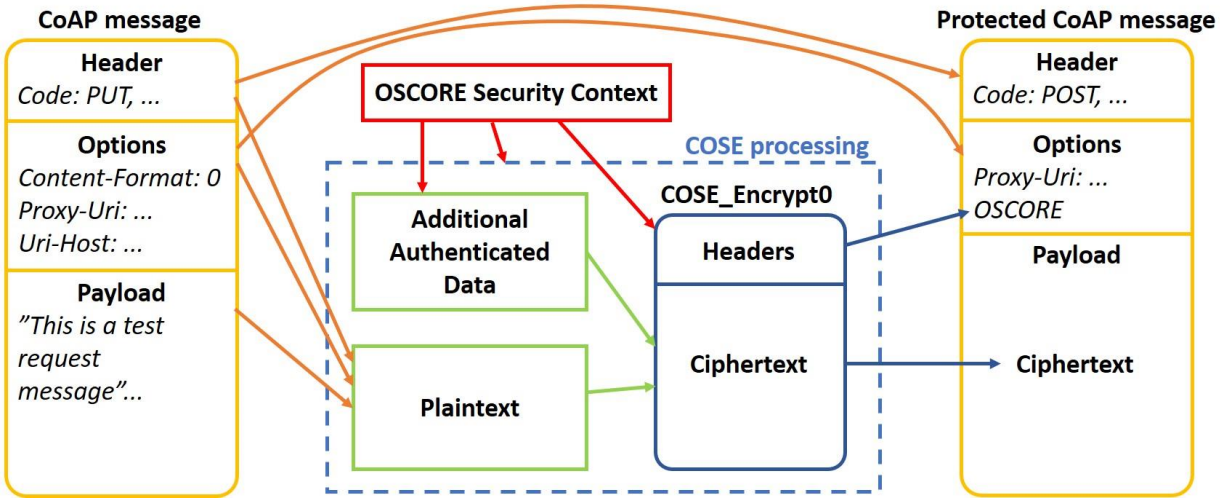

Figure 6 OSCORE protection

OSCORE provides end-to-end encryption, integrity protection, and replay protection of the CoAP messages. Replay protection is achieved using sequence numbers which also provide message ordering and binding between request and response messages. OSCORE uses the COSE message-type field for encrypted content to single recipients (COSE_Encrypt0, where the CEK is pre-shared). OSCORE encrypts with an Authenticated Encryption with Additional Data (AEAD) algorithm. One AEAD algorithm[4] is *mandatory to implement* to ensure interoperability among implementations. There is a

[4] AES-CCM with 128-bit encryption key and 64-bit (truncated) message authentication code

dedicated CoAP option defined called "OSCORE," which indicates OSCORE-based protection of the CoAP message. OSCORE can be used to secure multicast messages in group communications [MC-OSCORE].

An OSCORE-protected message is created in the following way (refer to Figure 6): The payload (if present) together with certain CoAP options and some of the header fields of the original unprotected CoAP message (e.g. Code) are encrypted and integrity protected as a COSE object. The other CoAP header fields and CoAP options are either integrity protected as Additional Authenticated Data or unprotected. The OSCORE option is neither encrypted nor integrity-protected as a whole, but some parts of the content such as key identifier, algorithm, and partial initialization vector are integrity-protected.

CoAP-to-CoAP forward proxies often handle translation between different transport layer protocols or addresses. OSCORE provides end-to-end security across such proxies, even though they are unaware of OSCORE. Restricted to subsets of HTTP and CoAP, OSCORE can also protect messages end-to-end from an HTTP endpoint to a CoAP endpoint across an HTTP-CoAP translation proxy (see [CoAP] and [HTTP-CoAP] for details on HTTP-CoAP translation) and the other way around. In this case OSCORE originates or terminates in an HTTP endpoint. The mapping between HTTP request and response messages and OSCORE-protected CoAP request and response messages is described in detail in [OSCORE]. A new header field, the HTTP OSCORE Header Field, is used for carrying the content of the CoAP OSCORE option when transporting OSCORE messages over HTTP hops.

OSCORE requires that client and server establish a security context with shared keying material to apply to the COSE objects protecting the CoAP messages. The security context also identifies other things such as the agreed AEAD algorithm, sender and receiver sequence numbers, and common initialization vector (IV). The unprotected header of the COSE object may contain information that identifies a security context. The shared keying material of such a security context can be obtained in different ways. The LwM2M v1.1 specification defines one solution in which the LwM2M bootstrap server configures a pre-shared key into the LwM2M device including other parts of the security context. The bootstrap server also provides the pre-shared key and other parts of the security context to the LwM2M server to which the LwM2M device is communicating. Other key management options are described in Section 5.

5. Key Management Aspects

OSCORE needs key management to establish protected communication. Key management and authorization to receive keys and privileges vary by service environment. Thus, different key-management systems may service LwM2M v1.1 to address the needs of a particular end-to-end security use case.

In every case, some key management system establishes the OSCORE security context between communicating parties. Each party keeps a security context consisting of a common context, sender context, and recipient context. The sender context is used when protecting messages to be sent and the recipient context is used when verifying received messages. The sender and recipient contexts are both derived from the common context that contains a shared key (a.k.a. master secret), information about AEAD and key derivation algorithms being used, common initialization vector, and a few optional values. One of the optional values is the ID Context which is a byte string used to identify the common context and provide additional information in the derivation of the AEAD keys of sender and recipient contexts.

The sender context contains an ID used to identify the sender, the AEAD key used for protecting OSCORE messages, and the sequence number used when sending messages. The sender sequence number is increased for each message sent. Similarly, the recipient context contains the recipient ID, the AEAD key used when verifying and decrypting received messages, and a replay window. The replay window denotes the set of sequence numbers that are acceptable for received messages.

5.1 Static Master Secret Throughout Device Life-Time

LwM2M v1.1 specifies an option in which the LwM2M bootstrap server configures the parameters needed by the LwM2M client on an IoT device to establish the security context. The master secret, sender ID, and recipient ID are mandatory parameters, and the rest of the parameters may rely on default values. Some of the non-mandatory parameters are frequently updated such as sequence number and replay window and typically stored in volatile memory. To allow for the possibility of using a static master secret throughout the life-time of the device, a LwM2M compliant implementation shall support a protocol (see Appendix B2 in [OSCORE]) for derivation of a new ID context. This ID context, used when deriving a new security context, ensures replay of previous messages are not accepted by the recipient in case non-mandatory parameters are lost due to unexpected reboot. The protocol is based on an OSCORE-protected exchange of two random nonces, generated by each party, where the concatenation of the two nonces constitutes the new ID context. The protocol is shown in Figure 7. It is required that the LwM2M endpoints supports a good source of randomness.

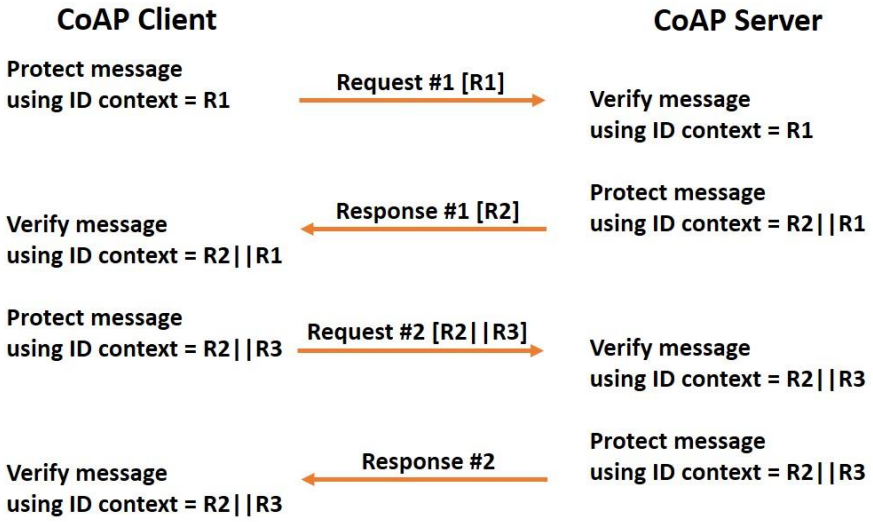

Figure 7 Establishment of new security context between two CoAP endpoints

For a very constrained device that is reporting only few bits per day, it is possible to use the same security context (same ID context is used) for more than 10 years still ensuring that replays of previously sent messages are not possible [OSCORE]. This requires careful updating of sequence number and replay window into non-volatile storage such that any unplanned reboot will not cause the same values being used again. A method achieving this is detailed in [OSCORE].

5.2 Retrieving the Master Secret From a Trusted Third Party

The establishment of a shared key can be accomplished by a third party that is trusted to vouch for an endpoint. The LwM2M bootstrap server provides one example of a trusted third party that may share the OSCORE master secret and other security context information between a LwM2M server that is trying to access a resource of a device (LwM2M client). A LwM2M bootstrap server authorizes the client to access a (frequently) shared key material and other information for establishment of new OSCORE security context between LwM2M clients and servers.

A similar example of a trusted third party is an Authorization Server (AS) that controls access to device resources in the Authentication and Authorization for Constrained Environments (ACE) framework [ACE-OAUTH]. A client that is trying to access a device resource must first retrieve an access token from the AS that can be presented to the device to prove that it is authorized to do so. When OSCORE is used to protect the communication between the client and the device, the access token contains the shared key (master secret) and other information such as algorithms that are needed to establish the OSCORE security context. The token is encrypted and integrity protected with keys shared between the device and AS. The access token, the shared key and other information that are needed to establish the context are securely delivered from the AS to the client using, for example, OSCORE or DTLS protected communication. In order to access a device resource, the client delivers the access token to the device. Random nonces, one from each party, are also exchanged. The client and the device may now compute the full security context in which the two nonces are used in the derivation as detailed in [OSCORE-ACE]. The client may then send a request to the resource of the device and receive a response in which both request and response are OSCORE protected using the derived security context.

5.3 Key Exchange Protocols for Derivation of a Fresh Master Secret

Automated key management through a key-management protocol derives a new fresh master secret for a new security context. A good key-exchange protocol offers forward secrecy and relies on a good source of randomness for keying material. Any type of automated key management comes with a cost in terms of, for example, extra power consumption for transmission of protocol messages and processing of the protocol that may not be negligible. On the other hand, a key exchange protocol allows an existing PKI to be used along with forward secrecy.

Here we consider two key exchange protocols for use with OSCORE. One is a COSE-based key exchange protocol called EDHOC, which is currently being defined within IETF, and with several reference implementations available. A second method uses the (D)TLS key exchange mechanism, which allows re-use of a (D)TLS implementation, if anyway available in the device. It would also interoperate with (D)TLS endpoints. Some data-security protocols have used DTLS or TLS Handshake Protocol to establish keying material.

5.3.1 EDHOC

The Ephemeral Diffie-Hellman Over COSE (EDHOC) draft specification [EDHOC] presents a COSE-based key exchange protocol. To provide forward secrecy, the two parties exchanging messages run a Diffie-Hellman (DH) key exchange protocol with ephemeral keys and derive shared keying material. The DH protocols are specified using COSE objects in which the DH key-exchange messages are authenticated using either pre-shared keys (PSK), raw public keys (RPK) or X.509 certificates. The credentials used in the authentication are established out-of-band, e.g. from a trusted third party such as a LwM2M bootstrap server.

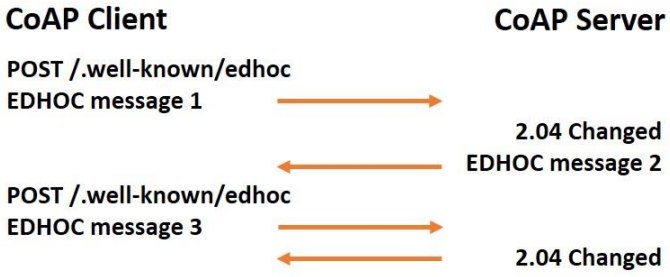

Figure 8 EDHOC protocol exchange between two CoAP endpoints

EDHOC is built on the three-message SIGMA (SIGn-and-MAc) protocol between two communicating parties. The EDHOC messages are encoded with CBOR and COSE with no requirements on lower layers: EDHOC messages could, for example,

be sent as CoAP message payloads. Thus, EDHOC protocol can be used between an HTTP endpoint and a CoAP endpoint across a middlebox performing HTTP-CoAP translation. The EDHOC protocol exchange between two CoAP endpoints is shown in Figure 8. The CoAP client initiates the EDHOC protocol at the CoAP server using the registered well-known URI "edhoc". The CoAP client makes a POST to the above mentioned URI with the first EDHOC message as payload. The response contains EDHOC message 2 in the payload and the CoAP client can then make a second POST to the same URI with EDHOC message 3 as the payload.

The EDHOC protocol is designed to be compact and lightweight in terms of message size and processing. EDHOC is not bound to a particular communication security protocol. It works off-the-shelf with OSCORE where it, in addition to establishing the master secret, also allows to negotiate the AEAD and key derivation functions used in OSCORE (and in EDHOC itself). Since EDHOC is built on the same protocols (CBOR and COSE) as OSCORE and even uses a common message format type (EDHOC messages uses the COSE_Encrypt0 message type with AEAD algorithm as used also in OSCORE) there is a large reuse of code and the total footprint for EDHOC + OSCORE can be kept small. As such, EDHOC can be used as authentication and key exchange protocol for establishing a derived key to use with COSE.

5.3.2 (D)TLS Handshake

Another option uses the key exchange mechanism of the well-known (D)TLS protocol to establish keys for OSCORE. In a constrained device that would anyway need to support (D)TLS this would be a way to minimize the device total footprint. Despite the name, Transport Layer Security and Datagram Transport Layer Security are not bound to the transport layer but can secure data at other layers of the protocol stack.

Several IETF draft specifications use the TLS or DTLS Handshake Protocol for constrained devices where CoAP is the bearer of (D)TLS handshake messages. An Internet Draft describes an architecture in which an application protects its data by directly using a TLS stack to generate and consume raw TLS records (from byte buffers) including a TLS handshake [ATLS]. The application sends TLS records to the recipient endpoint using a suitable protocol, e.g. HTTP, TCP, CoAP, etc. The architecture includes the setup for a TLS key exporter function, which after successful completion of the Handshake Protocol, may be used to export keys for use in another security protocol such as OSCORE. Other (expired) Internet Drafts proposed similar ideas earlier, such as [CODTLS] and [TLS-OSCORE].

A method is shown in Figure 9 for how to perform a TLS handshake on top of CoAP with the goal to create keying material for OSCORE [ATLS][TLS-OSCORE]. In this method the CoAP client requests TLS handshake records from its TLS implementation and sends them to a dedicated resource, called TLS-OSCORE in Figure 9, of the CoAP server. The CoAP server feeds its TLS implementation with the received records and receives new TLS records in return that are sent in response to the CoAP client. The client feeds them to its TLS implementation, receives new ones in return, and sends them to the TLS-OSCORE resource of the COAP server. This is repeated until the TLS handshake is completed and the TLS exporter is used for exporting the OSCORE master secret.[5]

[5] Note that the negotiation of COSE AEAD algorithm used by OSCORE is not described in these Internet Drafts.

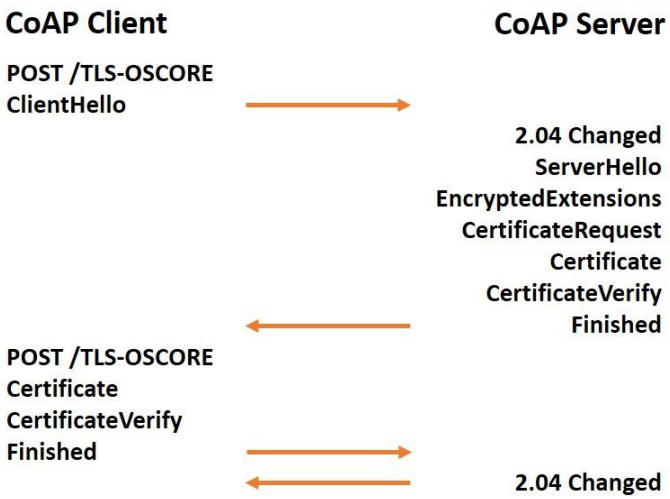

Figure 9 (D)TLS Handshake for key establishment for OSCORE

A device that hosts a LwM2M client typically has a TLS client implementation. In the (D)TLS Handshake method, it is the device that triggers the TLS handshake, and this works well with the LwM2M model where a LwM2M client connects to a LwM2M server. CoAP handles unreliable transports which means the LwM2M client can use a TLS handshake on top of CoAP instead of a DTLS handshake. There are cases where a DTLS handshake is preferred for its connection identifier feature, which the TLS Handshake Protocol lacks. But a DTLS handshake adds more message overhead than TLS handshake. EDHOC is optimized to reduce handshake overhead.

6. Conclusion

This paper analyzes end-to-end security requirements for IoT messages based on a security analysis, it presents three use cases that illustrate the challenges of end-to-end security, and it describes the LwM2M transport and application-security solutions for the use cases and security requirements.

Transport-layer Security is today commonly used to protect communication between two IoT endpoints. Although TLS provides end-to-end security for many IoT service topologies, it is not always sufficient to preserve integrity and confidentiality in topologies with proxies, gateways and other middleboxes for reasons described in Section 1 of this paper. Section 2 explains why application-layer security is needed for certain IoT service topologies where critical data are unprotected in middleboxes. Section 3 presents three use cases that need end-to-end application-layer security. Section 4 explains how LwM2M provides end-to-end security using COSE and OSCORE.

COSE is a message envelope format that's based on CBOR and thus designed for constrained devices. COSE-based OSCORE, part of LwM2M v1.1, protects each CoAP message. OSCORE provides an end-to-end security solution for use cases where CoAP runs across all hops of the IoT service path. This is true for the use cases “Explosive Gas Detection” and “Health Monitoring” of Section 3. These use cases feature OSCORE-unaware middleboxes that change addresses and network protocols, including non-IP protocols.

The “Protected Communication with the Web” use case, however, does not run CoAP end-to-end. In certain situations, OSCORE may still provide end-to-end security. One such example is shown in Figure 3 where we have LwM2M end-to-end between a LwM2M-aware application server and a LwM2M client, but both HTTP/TCP and CoAP/UDP transport messages across some portions of the service path. As long as messages have a one-to-one mapping between HTTP and CoAP, OSCORE provides end-to-end security across any HTTP-CoAP translation proxy.[6]

For the more generic case of connecting different IoT systems, illustrated by Figure 4, end-to-end security may be very hard to achieve. If the application layer protocol can be mapped to LwM2M/CoAP then the same procedure as in the previous case can be applied. If such a mapping is not possible then a dedicated application-security protocol using COSE as message envelope format of payload data may be used between the application server and the application in the IoT device. But then a number of conditions need to be considered, which are satisfied by OSCORE for CoAP. For example,

- any metadata that needs protection must be included in the payload (method, resource, data format),
- the application-security protocol protects against replay attacks, and
- the application-security protocol associates request and responses.

The application-security protocol may need to consider how protocol messages are mapped in the translation proxy.

Before OSCORE can be used, a security context must be established between the two communicating parties. LwM2M v1.1 provides the basic mechanisms in the LwM2M bootstrap server, which acts as a trusted third party for sharing key material. These basic mechanisms are sufficient in many cases and may be used in all the use cases of Section 3. Methods for deriving a new security context based on an existing shared master secret have been discussed in Section 5.

Establishing secure communication between a server and an IoT device often entails the right to access one or more resources of the IoT device. LwM2M may be combined with the ACE authorization framework for providing clients with rights to access resources of IoT devices. The ACE Authorization Server is another example of a trusted third party that may share a new master secret for use in establishing an OSCORE security context.

Besides the LwM2M methods for managing keys, Section 5 presents the ongoing standardization activities on key exchange protocols for use with OSCORE. These nascent standards allow for the establishment of a new fresh security context based on a new master secret. They also allow for operation over CoAP or HTTP. COSE-based EDHOC and TLS/DTLS handshake are two such key exchange protocols that satisfy all use cases in Section 3. They provide forward secrecy and authentication using either certificates, raw public keys, or pre-shared keys. The use of EDHOC as key establishment for OSCORE is currently the most suitable for constrained deployments of the key establishment methods outlined in this paper. There are efforts underway in the IETF to further optimize key establishment. Both EDHOC and TLS/DTLS handshake may be used as key establishment for other security protocols than OSCORE. For example, a COSE based approach to achieve

[6] In the case of Figure 3, the LwM2M server transforms a LwM2M message into an OSCORE-protected HTTP message by first converting the LwM2M message into a CoAP message, then protecting this message with OSCORE. This protected message is then mapped into an HTTP message following the mapping guidelines in [CoAP] and [HTTP-CoAP].

end-to-end security in the more generic case where LwM2M is not end-to-end may rely on either EDHOC or TLS/DTLS handshake for key establishment.

7. References

[ACE-OAUTH] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., Tschofenig, H., Authentication and Authorization for Constrained Environments (ACE), IETF Internet-Draft, 2019, Work in Progress, https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-24

[ATLS] Friel, O., Barnes, R., Pritkin, M., Tschofenig, H., Baugher, M., Application-Layer TLS, IETF Internet-Draft, 2019, Work in Progress, https://tools.ietf.org/html/draft-friel-tls-atls-02

[CBOR] Bormann, C., Hoffman, P., Concise Binary Object Representation (CBOR), https://tools.ietf.org/html/rfc7049

[CoAP] Shelby, Z., Hartke, K., Bormann, C., The Constrained Application Protocol (CoAP), https://tools.ietf.org/html/rfc7252

[CoAP-TCP] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., Raymor, B., CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets, https://tools.ietf.org/html/rfc8323

[CODTLS] Schmertmann, L., Hartke, K., Bormann, C., CoDTLS: DTLS handshakes over CoAP, IETF Internet Draft, 2014, Expired Work in Progress, https://tools.ietf.org/html/draft-schmertmann-dice-codtls-01

[Constantin] Constantin, L., Cisco starts patching devices against NSA-linked exploit, Infoworld, 26 August 2016, https://www.infoworld.com/article/3112947/firewall-software/cisco-starts-patching-firewall-devices-against-nsa-linked-exploit.html

[COSE] Schaad, J., CBOR Object Signing and Encryption (COSE), https://tools.ietf.org/html/rfc8152

[DTLS] Rescorla, E., Modadugu, N., Datagram Transport Layer Security Version 1.2, https://tools.ietf.org/html/rfc6347

[EDHOC] Selander,G., Mattson, J., Palombini, F., Ephemeral Diffie-Hellman Over COSE (EDHOC), IETF Internet-Draft, 2019, Work in Progress, https://tools.ietf.org/html/draft-selander-ace-cose-ecdhe-13

[GDPR] General Data Protection Regulation, https://gdpr-info.eu/

[HTTP-CoAP] Castellani, A., Loreto, S., Rahman, A., Fossati, T., Dijk, E., Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP), https://tools.ietf.org/html/rfc8075

[LwM2M_CIoT] Slovetskiy, S., Magadevan, P., Zhang, Y., Akhouri, S., Lightweight M2M 1.1: Managing Non-IP Devices in Cellular IoT Networks, 2018, https://www.omaspecworks.org/white-paper-lightweight-m2m-1-1/

[LwM2M_Core] OMASpecWorks, "Lightweight Machine to Machine Technical Specification: Core, Approved Version 1.1, OMA-TS-LightweightM2M_Core-V1_1-20180710-A", 10 Jul. 2018, URL: http://openmobilealliance.org/release/LightweightM2M/V1_1-20180710-A/

[LwM2M_Trans] OMASpecWorks, "Lightweight Machine to Machine Technical Specification: Transport Bindings, Approved Version 1.1, OMA-TS-LightweightM2M_Transport-V1_1-20180710-A", 10 Jul. 2018, URL: http://openmobilealliance.org/release/LightweightM2M/V1_1-20180710-A/

[Mirai] Mirai (Malware), Wikipedia, https://en.wikipedia.org/wiki/Mirai_(malware)

[MC-OSCORE] Tiloca, M., Selander, G., Palombini, F., Park, J., Group OSCORE - Secure Group Communication for CoAP, IETF Internet-Draft 2019, Work in Progress, https://tools.ietf.org/html/draft-ietf-core-oscore-groupcomm-04

[NarrowBandIoT] GSMA, NarrowBand-Internet of Things (NB-IoT), https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/

[OSCORE] Selander, G., Mattson, J., Palombini, F., Seitz, L., Object Security for Constrained RESTful Environments (OSCORE), https://tools.ietf.org/html/rfc8613

[OSCORE-ACE] Palombini, F., Seitz, L., Selander, G., Gunnarsson, M., OSCORE profile of the Authentication and Authorization for Constrained Environments Framework, IETF Internet-Draft, 2019, Work in Progress, https://tools.ietf.org/html/draft-ietf-ace-oscore-profile-07

[RFC7228] Bormann, C., Ersue, M., Keränen, A., Terminology for Constrained-Node Networks, https://tools.ietf.org/html/rfc7228

[Schneier] Schneier, B., Data is a Toxic Resource, Schneier on Security, https://www.schneier.com/blog/archives/2016/03/data_is_a_toxic.html

[TLS] Dierks, T., Rescorla, E., The Transport Layer Security (TLS) Protocol Version 1.2, https://tools.ietf.org/html/rfc5246

[TLS-OSCORE] Mattsson, J., Using Transport Layer Security (TLS) to Secure OSCORE, IETF Internet-Draft, 2017, Work in Progress, https://tools.ietf.org/html/draft-mattsson-ace-tls-oscore-00

8. Acknowledgements

This whitepaper is a work product of the Open Mobile Alliance IPSO working group.

Authors: The following persons contributed substantial written content to this document: Mark Baugher (Consultant), Benjamin Damm (Itron), and Per Ståhl (Ericsson).

Contributors: The following persons contributed valuable ideas and feedback that improved the content and quality of this document: Robert Cragie (Arm), Nicolas Damour (Sierra Wireless), Patrik Ekdahl (Ericsson), Matthew Gilmore (Itron), Jan Höller (Ericsson), Ari Keränen (Ericsson), Mojan Mohajer (U-blox), Francesca Palombini (Ericsson), Göran Selander (Ericsson), Padmakumar Subramani (Nokia), and Hannes Tschofenig (Arm).

Appendix A. Approved History (Informative)

Document Identifier	Date	Sections	Description
OMA-WP-e2e_Sec_IoT	24 Oct 2019	all	Status changed to Approved by IPSO WG ref: OMA-IPSO-2019-0107R01-INP_WP_e2e_Sec_IoT_for_Final_Approval