D3.3

Final Report on Network and System Security Solutions

WP3 – Network and System Security

SIFIS-Home

Secure Interoperable Full-Stack Internet of Things for Smart Home

Due date of deliverable: 30/06/2023
Actual submission date: 30/06/2023

Responsible partner: RISE
Editor: Marco Tiloca;
E-mail address: marco.tiloca@ri.se

30/06/2023
Version 1.00

Project co-funded by the European Commission within the Horizon 2020 Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

The SIFIS-Home Project is supported by funding under the Horizon 2020 Framework Program of the European Commission SU-ICT-02-2020 GA 952652

Authors: Marco Tiloca (RISE), Rikard Höglund (RISE), Göran Selander (Ericsson), Paolo Mori (CNR), Marco Rasori (CNR)

Approved by: Valerio Frascolla (Intel), Domenico De Guglielmo (DOMO)

Revision History

Version	Date	Name	Partner	Section Affected Comments
0.1	19/04/2023	Document created, with TOC, placeholders and instructions	RISE	All
0.2	24/04/2023	Purged items to exclude; added more placeholders; updated figures	RISE	All
0.3	30/04/2023	Updated executive summary and conclusion; removed mentioning of moot components; added Annex B	RISE	Executive summary; Section 6.1; Section 7; Annex B
0.4	08/05/2023	Revised introduction	RISE	Section 1
0.5	14/05/2023	Revised section 2; motivations for choosing CoAP and OSCORE	RISE	Sections 2, 3.1 and 3.7
0.51	15/05/2023	Revised and extended section on the OSCORE profile of ACE	RISE	Section 5.1
0.52	22/05/2023	Updated references	RISE	All
0.53	26/05/2023	Revised use of EDHOC in CoAP; revised details on OSCORE background	RISE	Sections 3.7 and 6.2.1
0.54	28/05/2023	Annex C on Demo 1	RISE	Annex C
0.55	30/05/2023	Revised description of EDHOC	RISE	Section 6.2
0.56	30/05/2023	Revised integration of UCON in ACE	CNR	Section 5.4
0.57	02/06/2023	Annex D on Demo 2	RISE	Annex D
0.6	04/06/2023	Revised section on notification of revoked access tokens	RISE	Section 5.2
0.7	04/06/2023	Revised section on key provisioning for Group OSCORE using ACE	RISE	Section 6.1
0.8	04/06/2023	Added section on experimental evaluation of Group OSCORE	RISE	Section 4.1.7
0.9	05/06/2023	Revised section on Group OSCORE	RISE	Section 4
0.91	13/06/2023	Processed review from	RISE	All

		DOMO		
0.92	26/06/2023	Processed review from Intel	RISE	All
1.0	29/06/2023	Ready to Submit	RISE	All

Executive Summary

This document is an outcome of WP3 "Network and System Security" and provides the final description of the network and system security solutions designed and developed in the SIFIS-Home project.

The documented security solutions include protocols, mechanisms, controls and tools, which can be grouped under the three following activity areas: i) Secure and Robust (Group) Communication; ii) Access and Usage Control for Server Resources; and iii) Establishment and Management of Keying Material.

These are intended to be applicable especially – but not only – to the use cases and IoT-based Smart Home environment considered in this project. Also, they are designed to be scalable as well as efficient and effective, thus limiting the impact on performance of the networks and applications involving also IoT devices.

The security solutions have been designed and developed consistently with the requirements presented in the deliverable D1.2 "Final Architecture Requirements Report", as well as with the SIFIS-Home architecture presented in the deliverable D1.4 “Final Component, Architecture, and Intercommunication Design”.

The security solutions presented in this document have been implemented, individually demonstrated through early focused demos, and finally integrated in the SIFIS-Home testbed within WP5 “Integration, Testing and Demonstration” as well as in the Smart-Home pilot within WP6 “Smart Home Pilot Use Case”.

Activities and results from WP3 have considerably contributed to dissemination activities in WP7 "Dissemination, Standardization and Exploitation", especially with academic publications in international venues and technical seminars, as well as with standardization activities in the Working Groups CoRE, ACE, LAKE and SCHC of the renowned international body Internet Engineering Task Force (IETF).

This document builds on, updates and obsoletes the previous deliverable D3.2 “Preliminary report on Network and System Security Solutions” from WP3, thus providing a self-contained, final description of the network and system security solutions developed during the project and integrated in the SIFIS-Home solution.

Table of contents

1 Introduction

This document is the third and final deliverable from the Work Package WP3 "Network and System Security", and provides the final description of the security solutions, mechanisms and approaches for network & system security developed in WP3 during the SIFIS-Home project and integrated into the SIFIS-Home solution.

A first description of the developed security solutions developed in WP3 up until March 2022 was provided in the deliverable D3.2 “Preliminary report on Network and System Security Solutions” [D3.2]. According to plans, the present document updates and obsoletes the previous deliverable D3.2. Therefore, the present document acts as a self-standing, final description of the network & system security solutions from WP3.

While displaying relations to one another, the different topics covered in WP3 can be mapped to three different activity areas, namely: i) Secure and Robust (Group) Communication; ii) Access and Usage Control for Server Resources; and iii) Establishment and Management of Keying Material.

Consistently with the organizational structure of WP3 and in order to closely reflect its activity areas, the core of this document includes one section for each activity area, with each of such sections presenting the security solutions pertaining to its corresponding area. According to this organization of content and consistently with the description of work, the contributions of this document have been organized as follows.

Section 2 provides a high-level overview of the network and system security solutions from WP3, highlighting the activity area and Task(s) where they have been carried out, as well as how the different security solutions relate and interact with one another. The detailed description in the following Sections 4, 5 and 6 additionally specifies how each security solution relates to the requirements defined in deliverable D1.2 “Final Architecture Requirements Report” [D1.2], as well as to the SIFIS-Home architecture components defined in deliverable D1.4 "Final Component, Architecture, and Intercommunication Design" [D1.4]. A further, at-a-glance overview of such mapping is also compiled as a single table, in Annex B of this document.

Section 3 introduces the main background concepts and technologies required to understand the security solutions presented in the following sections. These especially include: the Constrained Application Protocol (CoAP); the secure communication protocol Object Security for Constrained RESTful Environments (OSCORE); as well as the Authentication and Authorization for Constrained Environments (ACE) framework. This section is the only one providing background concepts and technologies, which were already fully available at the beginning of the project, and hence have been used as building blocks to develop novel security solutions based on and building on those. Conversely, all the content presented after this section concerns security solutions that have been developed in the SIFIS-Home project, and hence are in all effect foreground.

Section 4 considers the activities under the area "Secure and Robust (Group) Communication for the IoT", whose work has been done under Task T3.1 "Secure, interoperable and robust communication". This work has especially focused on the novel security protocol Group OSCORE, which enables end-to-end protected communication for IoT devices, also when based on a one-to-many group communication model and in the presence of (untrusted) transport intermediaries such as proxies.

Section 5 considers the activities under the area "Access and Usage Control for Server Resources", whose work has been jointly done under Tasks T3.2 "Security Lifecycle Management" and T3.3 "Dynamic Multi-Domain Security and Safety Policy Handling". This work has especially focused on methods to enforce fine-grained access control for resources at their hosting server contextually with end-to-end secure communication, and on techniques for dynamically evaluating access policies and consequently adjust/revoke stale access credentials.

The above includes the OSCORE profile of the ACE framework for end-to-end secure communication, and methods to achieve dynamic access and usage control of server resources, possibly within the ACE framework.

Section 6 considers the activities under the area "Establishment and Management of Keying Material", whose work has been done under Task T3.2 "Security Lifecycle Management". This work has especially focused on protocols and methods to distribute and establish keying material, as particularly intended for end-to-end protection of messages exchanged also in group communication environments. The above especially includes: using the ACE framework to distribute keying material for Group OSCORE upon the authorized joining of group members; and the EDHOC protocol for lightweight, authenticated establishment of OSCORE keying material among two peers, with an optimization allowing protected data to be exchanged after one round trip.

The security solutions presented in this document have been first implemented and individually demonstrated through early focused demos, which are described in Appendix C and Appendix D of this document. Such individual implementations are available as Open-Source Software, and have undergone continuous improvements throughout the second half of the project, as the related security solutions were revised.

Once the implementations mentioned above reached a sufficient level of stability and maturity, they have also been integrated into the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 “Integration, Testing and Demonstration” and into the Smart-Home pilot within WP6 “Smart Home Pilot Use Case”.
The actual integration is enabled by an integration-oriented component of the SIFIS-Home architecture also developed in WP3, namely the “CoAP Manager” component of the “NSSD Manager” module. This makes it possible to use the SIFIS-Home pub-sub system based on a Distributed Hash Table (DHT) for issuing commands to and obtain outcomes from CoAP client devices, and for allowing specific server devices to log events to a cloud instance of the IoT integration platform Yggio.

The actual integration into the SIFIS-Home solution relies on the process for continuous integration and deployment of Software used in the project, mostly based on Software containerization in multi-architecture Docker images and the automated execution of Github actions for building and deploying such images.
The integrated implementation of the security solutions is also available as Open-Source Software in the project Github organization, and has been regularly updated as the implementation of the individual security solutions were revised. Further details on the integration of the security solutions are provided in the deliverable D5.4 “Final version of the SIFIS-Home Security Architecture Implementation”.

Finally, the work and results from WP3 have substantially contributed to dissemination activities in WP7 "Dissemination, Standardization and Exploitation", with reference to academic publications in international venues and technical seminars, as well as to standardization activities in the Working Groups CoRE, ACE, LAKE and SCHC of the renowned international body Internet Engineering Task Force (IETF).

2 Overview of Network and System Security Solutions

The design and development of network & system security solutions in WP3 are carried out over three different activity areas, which are overviewed in Sections 2.1-2.3 and are put in relation with one another in Section 2.4.

For each developed security solution, a dedicated description is provided in the following Sections 4, 5 and 6, together with how the solution in question relates to the requirements defined in deliverable D1.2 “Final Architecture Requirements Report” [D1.2], as well as to the SIFIS-Home architecture components defined in deliverable D1.4 "Final Component, Architecture, and Intercommunication Design" [D1.4].

Figure 2.1 highlights how the security solutions developed in WP3 relate with the SIFIS-Home architecture. In

particular, they pertain to the components "Key Manager" (see the blue frame) and "Authentication Manager" (see the green frame) of the module "Secure Lifecycle Manager", as well as to the components "Content Distribution Manager" (see the red frame) and the "Secure Message Exchange Manager" (see the yellow frame) of the module "Secure Communication Layer".

The additional component "CoAP Manager" (see the purple frame) of the module "NSSD Manager" was also developed, in order to enable the effective integration of CoAP communicating devices into the SIFIS-Home solution, leveraging the DHT-based pub-sub system used in the project. Such integration aspects are discussed in the deliverable D5.4 “Final version of the SIFIS-Home Security Architecture Implementation” [D5.4].

Annex B of this document compiles a single table that provides a further, at-a-glance overview of how the WP3 security solutions are mapped against the requirements and the SIFIS-Home architectural components.

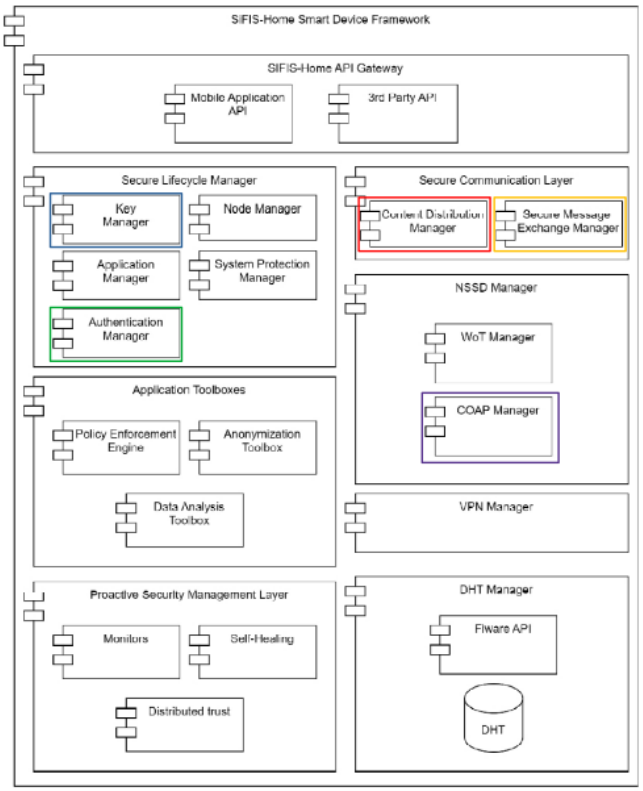


Figure 2.1 - Relation of WP3 Security Solutions with SIFIS-Home Architectural Components

Taking a higher-level perspective, Figure 2.2 positions the WP3 security solutions with respect to the SIFIS-Home architecture from a layer-based points of view, as well as with respect to the different Tasks of WP3.

Security solutions in the area “Secure and robust (group) communication for the IoT” have been developed in Task T3.1 “Secure, Interoperable and Robust communication” (see the red frame), hence contributing to the expected Secure Robust and Resilient Communication (see the black box), by ensuring lightweight secure communications among (IoT) devices in the Smart Home networked system, including exchange of network messages secured end-to-end.

Security solutions in the area “Access and Usage Control for Server Resources” have been developed in Task T3.2 “Security Lifecycle Management” (see the green frame) and Task T3.3 “Dynamic Multi-Domain Security and Safety Policy Handling” (see the purple frame). Also, security solutions in the area “Establishment and Management of Keying Material” have been developed in Task T3.2 “Security Lifecycle Management” (see the green frame). Such a set of security solutions contributes to the expected Secure Lifecycle Management (see the

light blue box), by ensuring security management in the Smart Home system through the device/system lifecycle, including fine-grained access control, and establishment/distribution/renewal of security credentials and keying material.

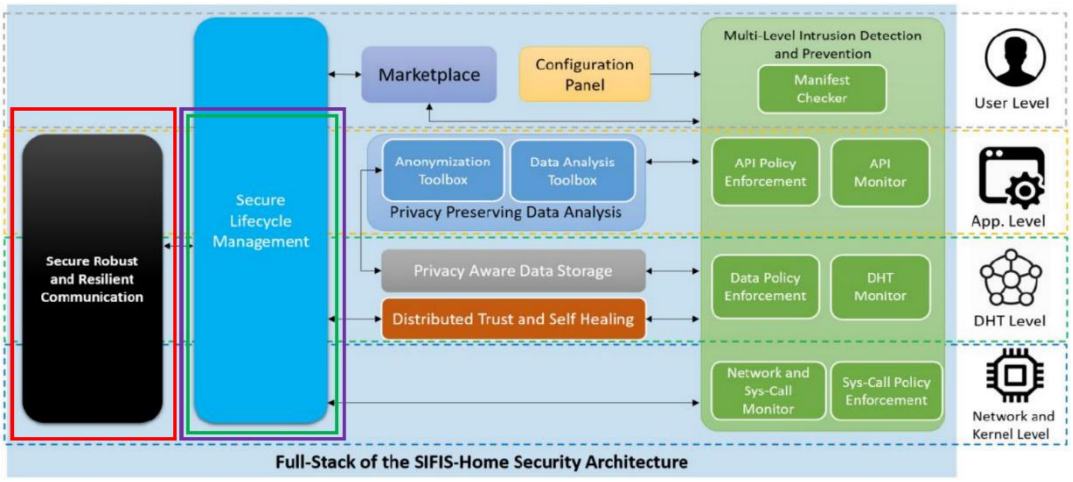


Figure 2.2 - Relation of WP3 Security Solutions with the SIFIS-Home Architecture

Secure and Robust (Group) Communication for the IoT

2.1 The work on this activity area was carried out within the Task T3.1 "Secure Interoperable and Robust Communication". The topics addressed in this activity area are presented in Section 4 and summarized below.

Group OSCORE – The security protocol Group Object Security for Constrained RESTful Environments (Group OSCORE) [TIL23i] has been developed to protect communications end-to-end when the Constrained Application Protocol (CoAP) [SHE14] is used in a group communication environment [RAH14][DIJ21]. That is, a CoAP client can send a request intended to multiple recipients (e.g., over IP multicast), each of which can reply with an individual response. Group OSCORE builds on the security protocol OSCORE [SEL19], by using the same core components CBOR [BOR20] and COSE [SCH22a][SCH22b], and provides end-to-end security of CoAP messages at the application layer. Group OSCORE provides source authentication of exchanged messages, and it ensures secure binding between a request and all the associated responses.

In particular, Group OSCORE provides two modes of operations, and each message exchanged in the group can be protected in either mode. The group mode is typically used for one-to-many messages (e.g., group requests over IP multicast) and ensures source authentication by means of a signature computed by the sender with its private key and embedded in the sent message. The pairwise mode is typically used for one-to-one messages (e.g., individual responses to a group request) and ensures source authentication without a signature – hence
2.2 considerably reducing communication overhead – by protecting a message with symmetric keying material derived from asymmetric authentication credentials of the two specific communicating peers.

Activities on this topic have especially built on the cooperation between RISE and Ericsson.

Access and Usage Control for Server Resources

The work on this activity area occurred within the Tasks T3.2 "Security Lifecycle Management" and T3.3 "Dynamic Multi-Domain Security and Safety Policy Handling". The topics addressed in this activity area are presented in Section 5 and summarized below.

OSCORE profiles of ACE – The ACE framework for authentication and authorization in constrained environments (ACE) [SEI22] delegates to separate specifications the details about secure communication

between the ACE entities, especially Clients and Resource Servers. Activities on this topic have defined the OSCORE profile of ACE, which enables secure communication between Client and Resource Servers as based on OSCORE. The profile provides mutual authentication of Client and Resource Server, as well as proof-of-possession of involved secret keys. It has been published as the IETF Proposed Standard RFC 9203.
Activities on this topic have especially built on the cooperation between RISE and Ericsson.

Notification of revoked access credentials – As authorization credentials, the ACE framework relies on Access Tokens, which may not only expire but also be early revoked. However, discovering about revoked Access Tokens is limited to ACE Resource Servers, through an actively started "introspection" procedure to be performed for one Access Token at the time. Activities on this topic have designed a solution to enable automatic and efficient notification of revoked, although non expired yet, Access Tokens to any device, supporting different levels of granularity in the reported information. This in turn can act as a building block to enforce usage control through the dynamic revocation of access credentials, following changes in the evaluation of access control policies.
Activities on this topic have especially built on the cooperation between RISE and Ericsson.

Usage control framework – The Usage Control (UCON) model encompasses and extends the traditional access control models introducing new features in the decision process: the mutability of attributes of subjects and objects and, consequently, the continuity of policy enforcement. The Usage Control System (UCS) is an extension of the well-known XACML standard. In particular, the XACML language has been enriched with new constructs to enable attribute mutability and to specify policy rules that need continuous enforcement. In addition, the XACML architecture has been extended with new components for keeping the current state of accesses, and to enable the continuous evaluation of ongoing accesses.

Combined enforcement of access and usage control – The ACE framework relies on an Authorization Server (AS) to issue access credentials in the form of Access Tokens. When doing so, the AS is agnostic of the exact approaches taken to evaluate access control policies for the different Clients requesting an Access Token. Also, if dynamic policy evaluation is used as a building block to enforce usage control, this would practically require convenient means to promptly notify about possible revoked Access Tokens. Activities on this topic have focused on bringing together some of the items mentioned above, in order to: i) perform advanced and dynamic evaluation of access control policies on the AS, by means of an advanced policy evaluation engine; and ii) enable the automatic notification of revoked access credentials. This practically enables the ACE framework to enforce access & usage control patterns for accessing the resources of the servers.
Activities on this topic have especially built on the cooperation between RISE and CNR.

2.3

Establishment and Management of Keying Material

The work on this activity area occurs within the Task T3.2 "Security Lifecycle Management". The topics addressed in this activity area are presented in Section 6 and summarized below.

Management of keying material for group OSCORE – Group communications for the CoAP protocol protected with the Group OSCORE security protocol rely on an OSCORE Group Manager acting as Key Distribution Center. Among other things, the Group Manager is mainly responsible for driving the joining process of new authorized group members. In case a joining node proves itself to be authorized to join the group, the Group Manager makes the joining node an effective group member and provides it with the required parameters and keying material to securely communicate in the group using Group OSCORE. Also, the Group Manager acts as trusted repository of public authentication credentials of the group members, which the Group Manager collects and provides during a joining process. Furthermore, the Group Manager provides possible assistance to current group members by means of a set of dedicated operations available to those.
Activities on this topic have especially built on the cooperation between RISE and Ericsson.

EDHOC - Key establishment for OSCORE, including adaptations and optimizations for CoAP – The authenticated key establishment protocol Ephemeral Diffie-Hellman Over COSE (EDHOC) has been developed to enable the lightweight establishment of keying material between two constrained devices, using COSE as its core building block. Key establishment through EDHOC also provides mutual authentication of the two devices and Forward Secrecy of the established keying material. Its main use case is the ultimate establishment of an OSCORE Security Context that the two devices can use to protect their communications with OSCORE. Specific adaptations and performance optimizations for CoAP have also been developed, especially the merging into a single message of, on the one hand, the last EDHOC message, and, on the other hand, the first request protected with the OSCORE Security Context derived through an EDHOC execution. This allows two devices to complete both key establishment and a first, protected data exchange in only two round trips in total.
Activities on this topic have especially built on the cooperation between RISE and Ericsson.

2.4 *Relation and Interaction Between the Security Solutions*

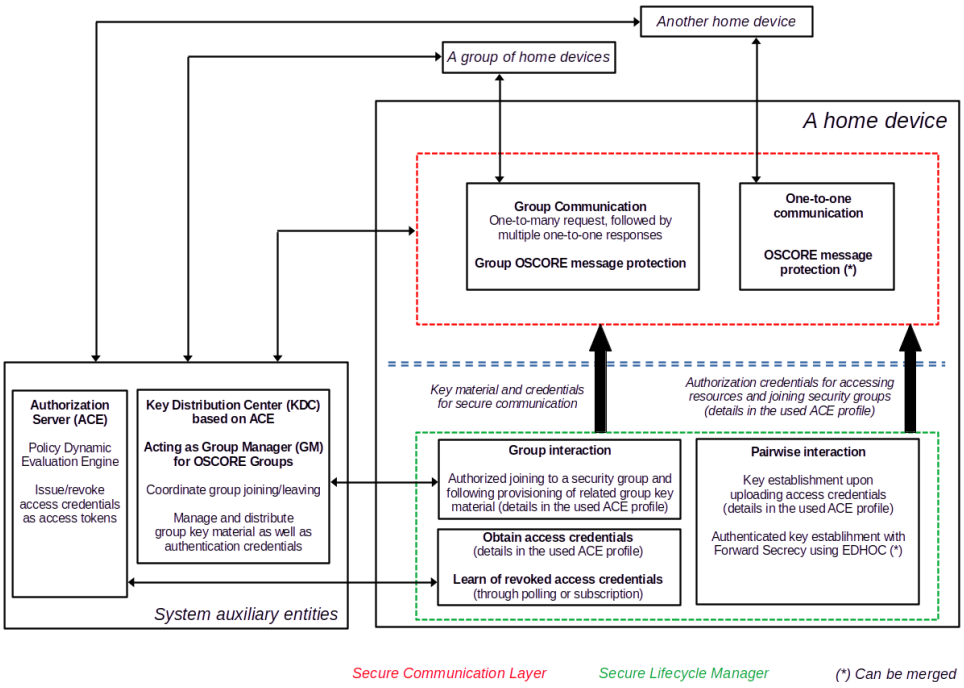


Figure 2.3 - Overview of the network & system security solutions and their relation

Figure 2.3 provides a graphical overview of the security solutions mentioned above, as well as of how they relate and interact with one another. Security solutions available to a home device as well as their domain can be fairly split into two, i.e., what lies above and below the double-dashed blue line.

What lies above the double-dashed blue line relates to secure and robust communication, i.e., to the first activity area summarized above, and pertains to the SIFIS-Home module “Secure Communication Layer”. In particular, it comprises OSCORE to provide end-to-end protection of one-to-one messages, as well as Group OSCORE to provide end-to-end protection of one-to-one and one-to-many messages in group communication exchanges.

What lies below the double-dashed blue line is related to the security lifecycle management, i.e., to the second and third activity areas summarized above. In particular, this comprises: i) the use and processing of access credentials issued by an external Authorization Server (AS), which can especially be used to enforce authorization in joining security groups where Group OSCORE is used; and ii) methods for establishing and renewing keying material between two endpoints (either directly or as part of an authorization workflow), as well as for obtaining

group keying material as part of an authorized group joining process.

Some security solutions related to security lifecycle management are facilitated by auxiliary entities, typically in control of a privileged Administrator. These are: i) the KDC responsible for one or multiple security groups; and ii) an AS responsible for issuing access credentials according to pre-installed access policies.

3 Background Concepts and Technologies

This section introduces background concepts and technologies referred to hereafter in the document.

This section is the only one providing background concepts and technologies, which were already fully available at the beginning of the project, and hence have been used as building blocks to develop novel security solutions based on and building on those. Conversely, all the content presented after this section concerns security solutions that have been developed in the SIFIS-Home project, and hence are in all effect foreground.

CoAP

The Constrained Application Protocol (CoAP) [SHE14] is an application layer, web transfer protocol based on
3.1 the Representational State Transfer (REST) paradigm [FIE00]. CoAP is designed for resource constrained devices and networks, and is now a de-facto standard application-layer protocol for the IoT.

CoAP typically runs on top of UDP [POS80], is not session-based and can handle loss or delayed delivery of messages. More recently, [BOR18] has defined how CoAP can work also on top of TCP [POS81] as well as WebSockets [FET11].

In typical resource-constrained deployments, several IoT nodes have limited resources in terms of memory, computing power, and energy (especially if battery-powered). This results in constrained network segments, e.g., due to lossy channels and limited bandwidth [BOR14]. In order to cope with this, resource-constrained nodes tend to adopt an asynchronous and intermittent communication model, i.e., they handle network traffic according to sending/receiving timeslots. To save energy, nodes can go offline (sleep), between two active timeslots, considerably extending their lifetime.

To manage these limitations, CoAP features an asynchronous messaging model and has native support for proxying. That is, proxies are used as intermediaries to enable access to server nodes that are not always online, by forwarding requests addressed to them, as well as caching and forwarding back their responses with consequent reduction of communication latencies.

Being a RESTful protocol, CoAP considers a Client and a Server as communicating parties, where the Client sends a Request to the Server, which replies by sending a Response. Depending on the operation to perform, CoAP Requests can be of different REST types, e.g., GET, PUT, POST, FETCH, PATCH/iPATCH and DELETE.

A CoAP message is divided into header and payload. The header is composed of: i) 4 bytes, including information such as the REST code specifying the different request/response type and a Message ID to match a Request/Response message with an Acknowledgment; ii) an optional Token to match request and response messages; and, possibly, iii) a number of options specified according to a Type-Length-Value format and used to control additional features. For example, CoAP options can be used to instruct a proxy on how to handle messages, specify for how long a message is valid, or indicate message fragmentation at the application layer.

A number of extensions for CoAP have been engineered. In particular, the Observe extension defined in [HAR15] allows a client to “subscribe” for updates to a resource representation at a server. That is, the client sends a first request targeting a resource at the server that it is interested in observing, including a new CoAP Observe option

in the request. Following a first response where the server confirms that an observation has indeed started, the server will send unsolicited responses, namely notifications, to the observing client, when the resource representation changes. All such notifications sent by the server will match the same original observation request.

Furthermore, taking advantage of the common use of the REST paradigm, CoAP messages can be mapped to HTTP messages [FIE22a][FIE22b][THO22] and vice versa, possibly with the aid of cross-proxies acting as translating intermediaries (see Section of [SHE14] as well as [CAS17]). This in turn has the benefit of enabling an effective integration of, on the one hand, IoT-based systems and networks composed of constrained devices, and, on the other hand, of non-constrained traditional web services using HTTP.

The original CoAP specification [SHE14] indicates only the Datagram Transport Layer Security (DTLS) 1.2 [RES12] protocol to secure message exchanges (see Section 3.4). More recently, the security protocol Object Security for Constrained RESTful Environments (OSCORE) [SEL19] has been standardized to provide end-to-end security of CoAP messages at the application layer (see Section 3.7).

The choice of CoAP as background technology to build on has been motivated by a number of reasons.

- CoAP is an ideal choice in the presence of resource-constrained devices and networks, since it is efficient in terms of processing, communication overhead, as well as state handling and retention.
- CoAP is an ideal choice for integrating IoT-based systems and networks with the web ecosystem based on HTTP, since it also relies on the REST paradigm and provides a bidirectional mapping with HTTP.
- CoAP natively provides a number of desirable features beyond the REST paradigm. These especially include Observation of resources, support for intermediary proxies, and caching of response messages.
- CoAP natively supports group communication, which allows for interacting with multiple devices at once. This enables efficient and low-latency communication to multiple endpoints, and is particularly suitable for controlling and monitoring appliances in scenarios like home and building automation.
- CoAP is used in lightweight industry standards, especially in the OMA Lightweight Machine-to-Machine (LwM2M) framework [OMA-CORE][OMA-TP] for monitoring and controlling IoT devices.
- CoAP is used in the following consortia, among others: Openthread; OCF; KXN IoT; OMA; 3GPP.
- CoAP is used by several manufacturers, including: Siemens; ARM; Assa Abloy; Qualcomm; Itron.
- CoAP is used in many operating systems, e.g.: Mbed; RIOT; Oniro; TinyOS; FreeRTOS; Contiki-NG.

3.2 *Group CoAP*

The CoAP protocol has been designed also to work in group communication scenarios, and in particular relying on IP Multicast. While the main CoAP specification [SHE14] describes the main features, this mode of operation has been detailed in the separate Experimental document [RAH14]. At the time of this writing, the new document [DIJ21] is intended to replace and obsolete [RAH14], by providing more up-to-date details concerning especially organization, maintenance and discovery of different types of groups; usage of CoAP Observation [HAR15] within groups; and security for group communication.

From a high-level point of view, the main feature of group communication is that a client may transmit a single request message as addressed to multiple recipient servers at once. Practically, this can be achieved by delivering the group request over IP Multicast. At this point, the client may not know the amount of other group members included in the group.

After that, all the servers listening to the IP Multicast address and receiving the single group request may reply to the client, each with its own individual response over IP unicast. This communication model is especially convenient for the following classes of applications.

- Discovery and identification of networked devices, or of particular services represented as resources at those devices. This relies on requests to interfaces for resource discovery, or on requests to well-known

resources hosted at reachable devices.

- Group controlling of multiple actuator devices, intended to synchronously act as a group, possibly with timing requirements. Typical examples include lighting applications or broadcast audio alert systems.
- Group status request from multiple devices, in order to monitor the status and operations of multiple units at once. After a first round of feedback, each device can be further polled and addressed individually through one-to-one communication.
- Network-wide queries and notifications, to broadly query or notify about status and change of specific information within a group of devices. In the case of network-wide notifications, a response is typically not expected to be sent back by the recipient servers.
- Software update distribution, in order to provide a same new software release or patch to multiple devices within a same group. Since large software updates are supposed to be transferred in smaller blocks, and due to the inherent unreliability of CoAP over multicast, backup mechanisms should be in place for servers to individually request for possible missing blocks.

When CoAP is used in a group communication scenario, a client can also send a request to the whole group over IP Multicast, for which it does not expect a corresponding acknowledgement back. In fact, this would be practically infeasible, since the client is not supposed to know how many servers are currently present in the group. This means that possible retransmissions of requests have to be handled by the client application, rather than by the actual CoAP stack layer.

Furthermore, the client sending a single group request has to be ready to receive multiple individual responses from the servers in the group, as matching to that same request. To this end, and unlike in one-to-one CoAP, the client has to preserve a group request beyond the reception of a first matching response, and until the expiration of a pre-defined timeout.

On the other hand, a server receiving a group request may either ignore it or send back a response to the client, depending on the application and its policies. If it replies, the server should not do that right after having processed the request, but rather after an additional randomly-selected time, up to a pre-defined leisure time. This makes it possible to avoid multiple servers simultaneously replying to a same group request, hence preventing message collisions and spreading responses over time in order to further prevent network congestion.

The original CoAP specification [SHE14] indicates only the Datagram Transport Layer Security (DTLS) 1.2 [RES12] protocol to secure message exchanges (see Section 3.4). However, DTLS is designed only to protect one-to-one message exchanges, over IP Unicast. Therefore, DTLS cannot be used to secure communications over IP Multicast in a CoAP group. To this end, the ongoing standardization proposal Group Object Security for Constrained RESTful Environment (Group OSCORE) [TIL23i] must be used as security protocol (see Section 3.3 4.1).

Channel Security and Object Security

Channel security refers to the transmission of data over a secure channel [RES03]. The secure channel can be negotiated at different layers of the protocol stack, i.e., at the data link, network or transport layer, through a specific establishment protocol. In particular, a secure channel handles data in an agnostic fashion, i.e., it has no knowledge of the secure data it conveys.

On the other hand, object security refers to protection mechanisms for data objects, and acts as an alternative to secure channels [RES03]. That is, instead of relying on a communication protocol at a lower layer to protect exchanged messages, applications also take care of protecting and verifying data objects of their own messages they generate and exchange.

DTLS

Datagram Transport Layer Security (DTLS) 1.2 [RES12] is an Internet standard providing channel security at the transport layer, to protect communications over unreliable datagram protocols such as UDP [POS80]. In particular, security is ensured hop-by-hop, between two nodes that are adjacent transport-layer hops. DTLS is a close copy of the Transport Layer Security (TLS) 1.2 protocol [DIE08] and provides equivalent security guarantees, i.e., it prevents tampering, eavesdropping and message forgery. Specifically, DTLS is adapted for use over UDP [POS80] instead of TCP [POS81], which is important for constrained devices and networks relying on
3.4 UDP as a connectionless transport protocol. The original CoAP specification [SHE14] indicates DTLS as the only security mechanism for protecting the exchange of CoAP messages.

Two communicating devices initially run the DTLS Handshake protocol, in order to exchange network- and security-related information, as well as to establish cryptographic keying material for later message protection. Specifically, one device acts as client and starts the Handshake execution, while the other device acts as server. The default Handshake mode relies on certificates, but constrained applications often prefer extensions based on symmetric pre-shared keys [ERO05] or on raw public keys [WOU14]. When the Handshake is completed and a secure session is established, the client and server can start exchanging data protected through the negotiated session keying material.

Currently, DTLS 1.3 [RES21] is an ongoing standardization proposal, aimed at providing the same security guarantees of the recently standardized TLS 1.3 [RES18] over unreliable transports such as UDP.

CBOR and COSE

3.5 Concise Binary Object Representation (CBOR) [BOR20] is a data encoding format designed to handle binary data. Its primary goal is achieving a very small parser code size, followed by the secondary goal of achieving a small message size.

CBOR Object Signing and Encryption (COSE) [SCH22a][SCH22b] builds on CBOR, and specifies how to perform encryption, signing and Message Authentication Code (MAC) operations on CBOR data, as well as how to encode the result in CBOR. As a further supporting feature, COSE defines how to encode cryptographic keys in CBOR.

3.6

End-to-End Security

A considerable amount of IoT devices is resource-constrained, with many of them even battery powered. Thus, it is important to limit their resource consumption, especially with respect to energy, in order to achieve a long device lifetime and acceptable performance. Improving energy performance may rely on device sleeping, which in turn results in the preference for an asynchronous communication model. However, to still provide well functioning communications and services, it becomes necessary to schedule requests to sleeping nodes with the help of proxies, used as intermediaries between clients and servers.

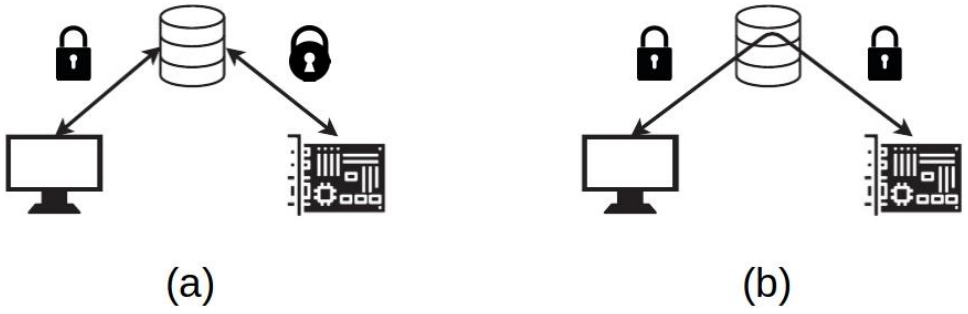


Figure 3.1 - Hop-by-hop vs. end-to-end security

The original CoAP specification [SHE14] indicates DTLS [RES12] as the only method to achieve secure communication for CoAP. As a result, when a proxy is deployed between a client and a server, messages are protected hop-by-hop, i.e., first between client and proxy, and then separately between proxy and server, as per the approach shown in Figure 3.1(a). Therefore, in the presence of an intermediary proxy, DTLS cannot provide end-to-end secure communication between a client and server node. Instead, a first secure channel has to be established between the client and the proxy, and a second, different secure channel has to be established between the proxy and the server. However, this results in a number of security and privacy issues and limitations.

First, this makes it necessary to perform a double security processing of CoAP messages, since the proxy has to decrypt a message received on the client DTLS channel, and then re-encrypt the same message for delivery on the server DTLS channel. This has an impact on the network and application performance, and especially on the Round Trip Time perceived by the client when eventually receiving a response.

Second, the proxy has to be necessarily trusted beyond what is strictly necessary to perform the intended operation. In fact, the proxy is effectively able to fully access the CoAP messages it relays to the server and back to the client. However, mandating such an extent of trust in proxies and in the service providers operating them clearly results in unnecessary and excessive exposure of data, which in turn creates opportunities to tamper with them and easily raises privacy implications.

Instead, Figure 3.1(b) shows an alternative approach based on end-to-end security, where the client and the server rely on a two-way secure communication context. This approach essentially consists in tunneling channel security through the proxy, and thus successfully overcomes the two limitations discussed above. However, to be practically deployable and functional, a solution based on end-to-end security must not prevent the proxy to correctly perform its intended operations, especially the forwarding of CoAP requests and the caching of CoAP responses. As a consequence, it must also be possible to selectively protect different parts of a same CoAP message in different ways, i.e., some encrypted, others only integrity protected and finally some parts fully accessible by the proxy.

This flexibility can be achieved by using object security, so that applications can choose which parts of an outgoing message have to be integrity-protected, encrypted, or both. It is worth noting that protecting only the CoAP payload is not sufficient, as it does not protect against several other attacks, such as changing the REST Code field in the CoAP header, e.g., from GET to DELETE, which tricks a server into deleting a resource instead of returning its representation.

The points discussed above have motivated the need for lightweight end-to-end security, which also preserves
3.7 proxying functionalities. This has in turn led to the design of OSCORE [SEL19], an application-layer protocol based on object security, which indeed fulfills these requirements.

OSCORE

This section describes Object Security for Constrained RESTful Environments (OSCORE). For the reader's convenience and due to space constraints, we only present the main features, while a complete description is available at [SEL19]. OSCORE provides message confidentiality, integrity and reordering/replay protection, as well as a weak freshness protection through sequence numbers for CoAP messages. To this end, OSCORE transforms an unprotected CoAP message into a protected CoAP message. A protected CoAP message includes the newly defined OSCORE option [SEL19], which signals the usage of OSCORE to protect the message, as well as an encrypted COSE object [SCH22a][SCH22b] in the CoAP payload.

OSCORE is designed for providing end-to-end security between two CoAP endpoints, while preventing intermediaries to alter or access any message field that is not related to their intended operations. The security concerns not only the actual payload of the original CoAP message, but also all the fully protected CoAP options, the original request and response REST code, as well as parts of the URI to resources targeted in request messages (see Section 3.7.3).

To be able to use OSCORE, the following two criteria must be fulfilled. First, the two CoAP endpoints are required to support CBOR and COSE (see Section 3.5), as well as the specific HMAC-based Key Derivation Function (HKDF) and Authenticated Encryption with Associated Data (AEAD) algorithms they want to use for key derivation and authenticated encryption, respectively. This assumption is often already fulfilled in the vast majority of IoT applications using CoAP. Second, the two CoAP endpoints are required to have an OSCORE security context (see Section 3.7.1), or the necessary information and keying material to derive it. While this has to happen in a secure and authenticated way, and some suitable approaches are proposed in [PAL22] (see Section 5.1) and [SEL23b] (see Section 6.3), OSCORE is not tied to any particular approach for context establishment.

A Java implementation of OSCORE from RISE has been integrated in the Californium library [CALIFORNIUM] from the Eclipse Foundation, and is available for use in the SIFIS-Home project. A Contiki-NG implementation of OSCORE from RISE is available at [OSC-DEV], and is intended to be integrated in the Contiki-NG operating system [Contiki-NG].

An experimental performance evaluation of the OSCORE protocol has been performed and published in [GUN21], based on the two implementations above and involving real resource-constrained IoT devices.

Consistent with the choice of CoAP as background technology (see Section 3.1), it is natural to choose OSCORE as the security protocol to protect CoAP communications and as further background technology to build on. In particular, OSCORE is targeted, as it provides end-to-end security of CoAP message exchanges across intermediary proxies, which is otherwise not achieved if using transport layer security (see Section 3.6).

Furthermore, OSCORE was designed as future-ready for allowing the protection of one-to-many requests and corresponding unicast responses. That is, it set the ground for being extended to a group communication setting, which has eventually led to the Group OSCORE security protocol developed in the project (see Section 4.1).

Finally, OSCORE is used as application-layer security protocol for CoAP in lightweight IoT frameworks, such as the OMA LwM2M framework [OMA-CORE][OMA-TP], and it is increasingly used in consortia and platforms that use CoAP altogether.

3.7.1 OSCORE Security Context

OSCORE uses parameters and keying material included in an OSCORE security context, and used to perform encryption and integrity protection operations. For this reason, every pair of communication endpoints, i.e., a CoAP client and CoAP server, share the same security context.

The security context consists of three parts: a Sender part, a Recipient part and a Common part. The Sender part is used to protect outgoing messages (i.e., requests on the client and responses on the server). The Recipient part is used to verify incoming protected messages (i.e., requests on the server and responses on the client). Finally, the Common part contains shared data. This division is illustrated in Figure 3.2.

An instance of a security context is present as a copy on the client and server, containing the same data values. However, as can be seen in Figure 3.2, the sender and recipient parts are mirrored, so that the sender part of the server corresponds to the recipient part of the client, and vice versa.

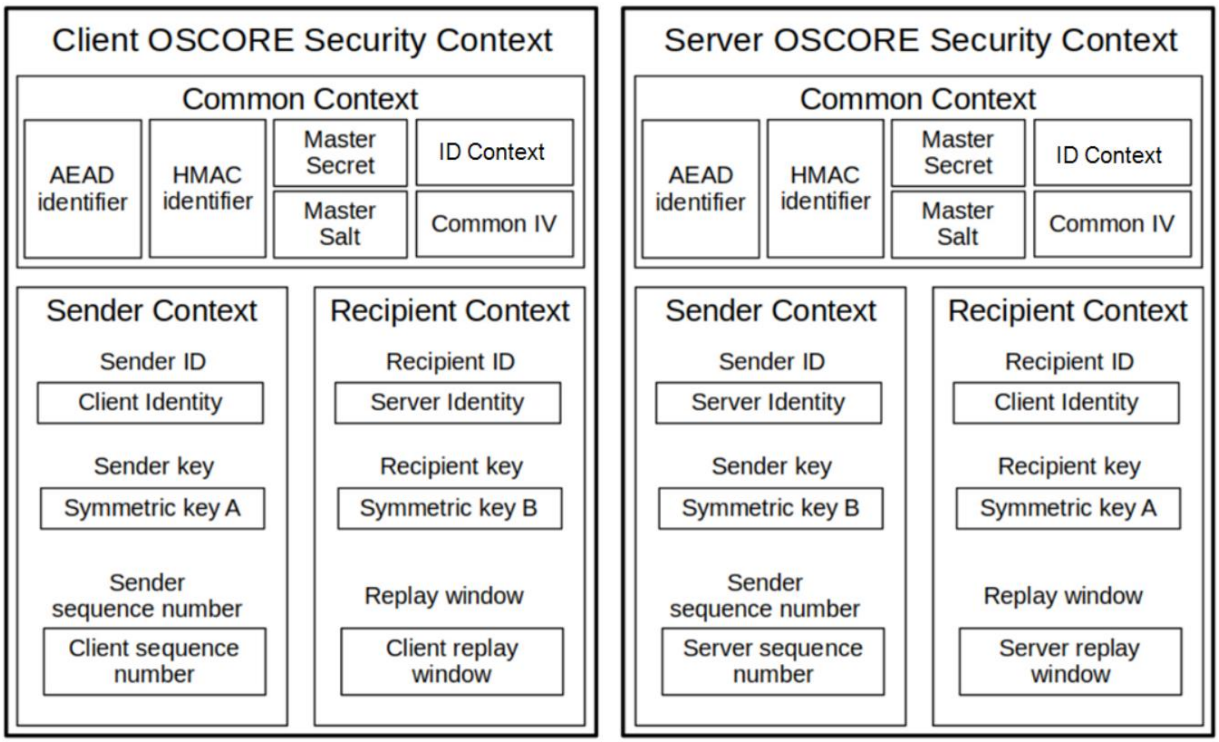


Figure 3.2 - OSCORE Security Contexts for a Client and Server pair (only the fields used during operation)

More in detail, the Common part includes:

- an identifier of the AEAD algorithm used to encrypt and authenticate exchanged messages;
- an identifier of the HMAC-based key derivation function used to derive keys and initialization vectors (IVs);
- the Master Secret: a random byte string used to derive keys and IVs;
- an optional Master Salt: a byte string used with the Master Secret to derive the keys and IVs;
- an optional ID Context: used to identify the Common Context and to derive keys and IVs;
- a Common IV to generate AEAD nonces. This is the only element of the common part which is going to be updated, throughout the lifetime of the Security Context.

The Sender part includes:

- a Sender ID, a byte string identifying the Sender part of the security context;
- a Sender Key, the symmetric key to protect outgoing messages;
- a Sequence Number, used for nonce generation to protect outgoing messages, and for replay protection of incoming messages (see Section 3.7.4).

The Recipient part includes:

- a Recipient ID, a byte string identifying the Recipient part of the security context;
- a Recipient Key, the symmetric key to decrypt incoming messages;

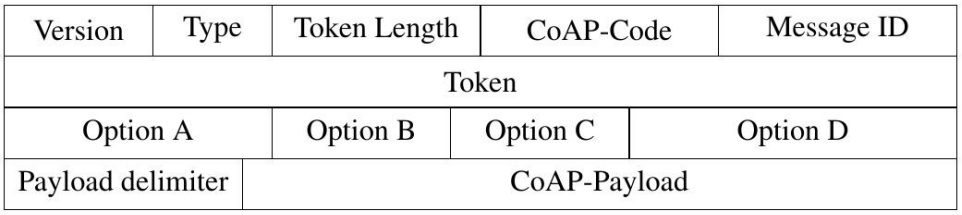
- a Replay Window to verify freshness of incoming messages on the CoAP server (see Section 3.7.4).

The combination of ID Context, Sender ID, Master Secret and Master Salt must be unique for each communicating pair of Client and Server. This ensures unique keys and nonces for the AEAD. Further details on establishing Sender/Recipient IDs and ensuring their uniqueness are out scope for OSCORE and this document.

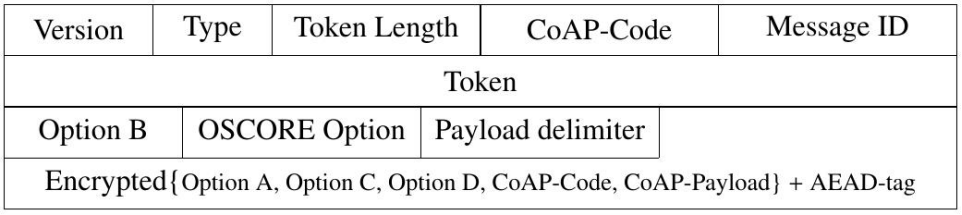
3.7.2 Protecting the CoAP Message

Different parts in a CoAP message are protected in different ways. That is, Confidential data, which should neither be read nor altered by a proxy, are both encrypted and integrity protected. Static data, which should be readable but not changed, are integrity protected but not encrypted. Dynamic data, which a proxy should be able to modify, are not protected. Finally, there are also Mutually known data, which the sender and receiver have agreed upon before exchanging messages. These data are part of the input to the integrity protection process, to ensure that the two communicating endpoints behave correctly and possibly detect anomalies. However, they are never sent as both parties already know them.

Figure 3.3 shows a comparison between an unprotected CoAP message and the resulting OSCORE-protected CoAP message. We can see that sensitive parts of a message are encrypted, e.g., some options and the payload, while others are left unencrypted, e.g., some options and some fields of the CoAP header. The encrypted content is placed into the payload of the protected message.



(a) CoAP message format.



(b) OSCORE message format.

Figure 3.3 - Message layout for unprotected and protected CoAP messages

The actual protection process takes as input an unprotected CoAP message and produces a protected OSCORE message as follows.

1. The confidential data are enclosed into a COSE object [SCH22a][SCH22b]. These include the REST code of the original CoAP message, a subset of the CoAP options, and the CoAP payload (if present). The CoAP options considered at this step are the ones not relevant for operations of intermediary (proxy) units.
2. The static fields of the CoAP header and static proxy-readable CoAP options need to be authenticated

and integrity protected, but not to be encrypted. This set of data composes the Additional Authenticated Data (AAD).

3. The COSE object is finalized, by encrypting and integrity protecting the data it encloses, while only integrity protecting the AAD. To this end, the Sender Key and the Sender Sequence Number from the Sender Context are used. The resulting ciphertext and AEAD-tag is included in the Message Content field of the COSE Object.
4. The COSE object is transported in the OSCORE-protected CoAP message. In particular, the COSE ciphertext is transported as payload of the OSCORE-protected message, while other information is encoded in the value of the OSCORE option. This especially includes the Sender Sequence Number used for message encryption ('Partial IV' field), the Sender ID of the sender endpoint ('kid' field), and the ID Context of the Security Context ('kid_context' field). Furthermore, any encrypted options are removed from the CoAP message. Finally, the original REST code is replaced with either POST (2.04) for a CoAP request (response), or with FETCH (2.05) for a CoAP request (response) using the CoAP mechanism Observe [HAR15] for which the POST method is not defined.

An analogous reverse process is performed upon receiving a protected message, together with anti-replay checks (see Section 3.7.4). To decrypt the protected message, the recipient CoAP endpoint uses the Recipient Key from its own Recipient Context associated with the message originator.

3.7.3 Proxy Functionalities and Data Protection

Building on the previous sections, we can now describe how OSCORE handles proxying of encrypted messages. OSCORE is designed to uniquely bind each request to the corresponding response, thus preventing proxies from serving cached responses to clients different from the one originating the request.

As previously stated, OSCORE cannot encrypt entire CoAP messages. An example of static data in a CoAP message which cannot be encrypted but should be integrity protected is the Version field of the CoAP header. This field has to remain readable, so that the receiver endpoint knows how to process an incoming message, but should be integrity protected to prevent future version-based attacks. The Token field of the CoAP header also has to remain readable, as it is used for binding each request to the corresponding response. However, unlike the Version field, the Token field cannot be integrity protected, as it can be modified by proxies, when a message traverses the network.

3.7.4 Replay Protection

OSCORE provides protection against replay and message reordering attacks. To this end, both the client and server store a sequence number and a replay window as part of the security context (see Section 3.7.1) and include said sequence number in every outgoing request, before incrementing it by 1.

Upon receiving a protected request, the server verifies that the conveyed sequence number was not received before. To correctly handle messages received out of order, OSCORE relies on a sliding window of sequence
3.8 numbers, where the server accepts only messages with sequence number greater than the lower bound of the replay window. In such a case, the server updates its replay window accordingly. Otherwise, the server considers the message to be a retransmission and discards it.

ACE Framework for Authentication and Authorization

Authorization can be described as the process for granting approval to an entity to access a resource. In practice, the authorization consists in enabling a requesting device to access a resource at a given host device.

IoT devices can be quite diverse in terms of available computing resources and communication capabilities which

means that there is a need to support many different authorization use cases. Furthermore, many IoT devices are constrained in terms of available memory, battery power, processing capabilities, network bandwidth, or some other resource. Thus, authorization solutions applied for IoT scenarios have to be flexible and feasible also for resource-constrained platforms.

In networks composed of IoT devices, most of the devices can in fact be constrained, with the notable exception of those that serve as intermediary proxies, key distribution servers or providers of some other central management service. Due to this reason, it can be beneficial to entirely offload decision making, authorization-related cryptographic operations and similar from the (constrained) host devices to a dedicated central management service. This is accomplished by separating authorization to access a resource from the actual resource itself. Additionally, it is convenient to centrally manage the granting to resource access in a network. This is especially beneficial when designing, managing, and operating the network.

The ACE framework for Authentication and Authorization in Constrained Environments [SEI22] is based on the widely deployed OAuth 2.0 [HAR12] authorization framework, and essentially enables its functionalities in the IoT. In particular, the ACE framework mainly relies on the following set of existing components.

The main functionality and overall approach are inherited from the OAuth 2.0 framework, a standard, widely adopted solution for authorization and access control. Another component is COSE [SCH22a][SCH22b], a compact encoding format for security information based on CBOR [BOR20], which is in turn a binary encoding format designed for small message sizes and code. Furthermore, the lightweight, web-transfer protocol CoAP [SHE14] is (preferably) used. In particular, CoAP can be used for communication scenarios where HTTP is not appropriate. In addition, CoAP typically runs on top of UDP [POS80], which provides additional benefits in the form of reduced message exchanges and overhead. Lastly, CoAP messages can be secured at the transport layer by using the DTLS protocol suite [RES12], and/or end-to-end at the application layer by using the OSCORE security protocol [SEL19].

By choosing well established components, existing authorization services can be brought into the IoT environment in a secure way. Additionally, these components are flexible enough to provide security for a wide range of IoT devices and deployments. This is important as IoT devices are quite diverse and can encompass both very resource constrained, battery-powered devices but also more capable devices with a reliable power supply.

A Java implementation of the ACE framework from RISE is accessible at [ACE-DEV], as available for use in the SIFIS-Home project.

3.8.1 ACE Entities

The units involved in a typical interaction as defined by the ACE framework are the following.

The Client (C) is a device wanting access to a specific resource at a given host, namely the Resource Server (RS). The Authorization Server (AS) is responsible for authorizing client devices to access remote resources hosted at a Resource Server, and for providing them with evidence of such authorization.

This evidence typically consists of an Access Token, and is used by C to access protected resources on the RS. Typically, an Access Token is represented as a CBOR Web Token (CWT) [JON18][JON20] efficiently encoded as an object in CBOR [BOR20], or alternatively as a JSON Web Token (JWT) [JON15][JON16] encoded in JSON [BRA17].

Detailed documentation on how ACE should be implemented for different scenarios can be found in related application and security profiles (see Section 3.8.3). In particular, the ACE framework delegates to the profiles the description of how to use the main specification with concrete transport and communication security protocols between the involved entities.

The following describes a typical interaction in the ACE framework between the involved entities C, AS and RS.

3.8.2 ACE Workflow

As also shown in Figure 3.4, the following steps occur during a full ACE transaction.

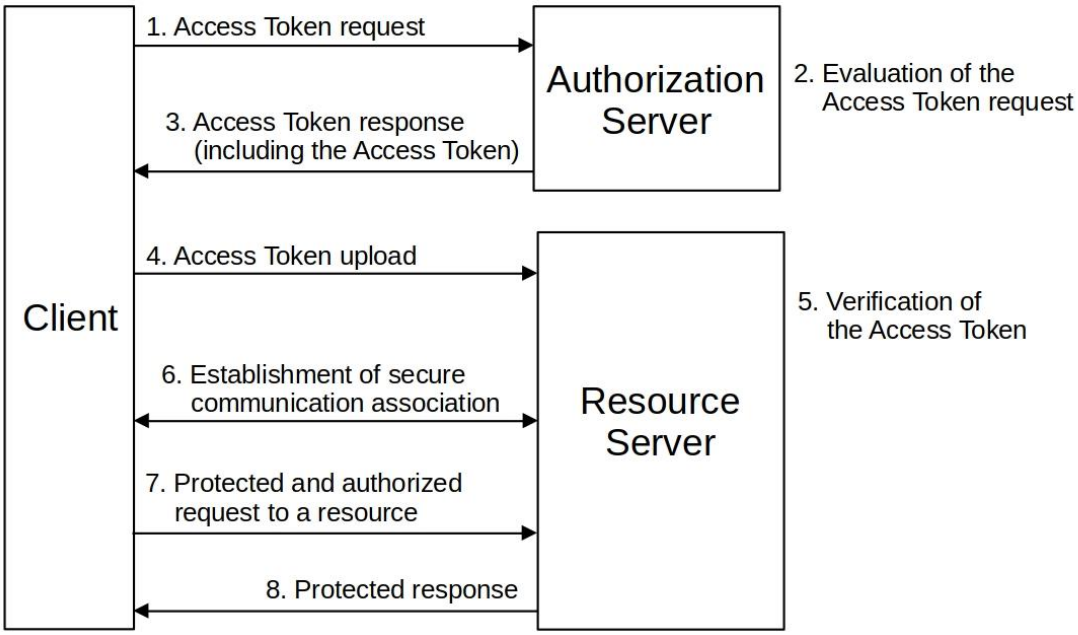


Figure 3.4 - ACE Framework workflow.

1. C sends a request for an Access Token to the AS, targeting the **/token** endpoint at the AS. When doing so, C specifies:
   - The target RS as "audience".
   - The requested “scope”, i.e., the resources it wishes to access at the RS and through which operations.
   - According to the used ACE profile and its selected mode, its own public key.

2. The AS evaluates the request from C against access control policies for the RS, as pre-established by the resource owner. In case of success, the AS produces an Access Token as evidence of the granted authorization. Depending on the used ACE profile and its mode, the AS also generates a symmetric key K, intended to be shared between C and RS. Then, the AS includes, among other elements, the following information in the Access Token:
   - The "audience" and the "scope" granted to C.
   - Optionally, an indication of the used ACE profile.
   - According to the used ACE profile and its selected mode, the symmetric key K or the public key of C.

3. The AS provides C with the following information:
   - The exact "scope" actually granted to C and specified in the Access Token, if it was possible to only partially satisfy the request.
   - Either the newly generated symmetric key K or, the public key of the RS, depending on the used ACE profile and its selected mode.
   - Optionally, an indication of the profile of ACE to use.
   - The Access Token, as either encrypted and authenticated, or instead signed. If encrypted, the Access Token is protected with keying material shared only between the AS and the RS. In either case, the

Access Token is opaque to C, which does not understand its content and structure.

4. In case of positive response from the AS, then C uploads the Access Token to the RS. This typically happens by sending a request to the **/authz-info** endpoint at the RS.

5. The RS verifies that the Access Token is intact and actually originated by the AS, by possibly decrypting it. Then, the RS verifies that the content of the Access Token is consistent with its own resources and scopes. If so, the RS stores the Access Token.

6. Depending on the used profile of ACE and its selected mode, C and RS perform possible additional exchanges and operations, in order to establish a secure communication association. To this end, they rely on the keying material facilitated by the AS during the previous steps, i.e., each other's public keys or the symmetric key K. Also depending on the used profile of ACE, parts of this step might be combined with the uploading of the Access Token at step 4.

7. C sends a request to RS, in order to perform an operation at one of the resources hosted at RS, consistently with the "scope" granted by the AS at step 3 above. The request is protected using the established secure communication association.

8. The RS checks the request against the Access Token stored for C, and verifies that the requested access and specific operation are in fact consistent with the "scope" in the stored Access Token. In case of positive outcome, the RS processes the request from C and possibly replies with a response. The response is protected using the established secure communication association.

As an optional feature, the AS can provide an additional service to the RS called "introspection". That is, upon receiving an Access Token or later on while storing it, the RS can send a request to the **/introspect** endpoint of the AS, specifying the whole Access Token or a reference to it. The AS can then return fresh information on the current status and validity of the Access Token, that the RS can consider to determine whether to accept or preserve the Access Token.

3.8.3 ACE Security Profiles

The following steps occur during a full ACE transaction.

Among other things, an ACE profile has to specify the following points.

- The Communication and security protocol for interactions between the involved entities, as providing encryption, integrity protection, replay protection and a binding between requests and responses.
- The method used by the client and Resource Server to mutually authenticate.
- The (secure) method for the client to upload an Access Token at the Resource Server.
- The specific key types used (e.g., symmetric/asymmetric), and the protocol for the Resource Server to assert that the Client possesses such keys (proof-of-possession).

The following Section 3.8.3.1 provides an overview of the DTLS profile of ACE. Further security profiles of ACE developed within the SIFIS-Home project are presented in Section 5.1.

3.8.3.1 DTLS Profile

The DTLS profile of ACE defined at [GER22] describes how a Client (C) and a Resource Server (RS) can engage in the ACE workflow and establish a DTLS channel for securely communicating with one another using the DTLS 1.2 protocol suite [RES12]. The DTLS profile provides two different modes of operation, i.e., the asymmetric mode and the symmetric mode.

Asymmetric mode. In the Token request to the Authorization Server (AS), C includes also its own public key. Then, the AS includes the public key of C into the Access Token to be released. After that, the AS provides C with both the Access Token and the public key of the RS. For the sake of proof-of-possession, C has to prove to the RS to possess the private key corresponding to the public key specified in the Access Token. After uploading the Access Token to the RS, C starts performing a DTLS handshake with the RS, in asymmetric mode. In particular, C and RS authenticate each other using their own public keys as Raw Public Keys (RPK) [WOU14].

Symmetric mode. Upon receiving the Token request from C, the AS generates a symmetric key K and includes it into the Access Token to be released. After that, the AS provides C with both the Access Token and the key K. For the sake of proof-of-possession, C has to prove to the RS to also possess the key K specified in the Access Token. After uploading the Access Token to the RS, C starts performing a DTLS handshake with the RS, in symmetric mode. In particular, C and RS authenticate each other using K as pre-shared key [ERO05]. As a particular, optimized alternative, it is possible to convey the whole Access Token itself within one of the handshake messages from C to the RS, rather than as a separate message before the handshake can start.

In both modes, a successful completion of the DTLS handshake achieves proof-of-possession. From then on, C and RS securely communicate using the established DTLS channel.

The Java implementation of the ACE framework from RISE accessible at [ACE-DEV] provides also the DTLS profile, both in asymmetric and symmetric mode, and is available for use in the SIFIS-Home project.

Dynamic evaluation of access policies

3.9 The Usage Control (UCON) model defined in [PAR04][ZHA05] encompasses and extends other traditional access control models. In particular, the UCON model introduces new features in the decision process, such as the mutability of attributes of subjects and objects and, consequently, the continuity of policy enforcement.

These features are meant to guarantee that the right of a subject to use a resource holds not only at access request time, but also while the access is in progress. As a matter of fact, rights are dynamic because subjects' and objects' attributes change over time, thus requiring continuous enforcement of the security policy during the access time.

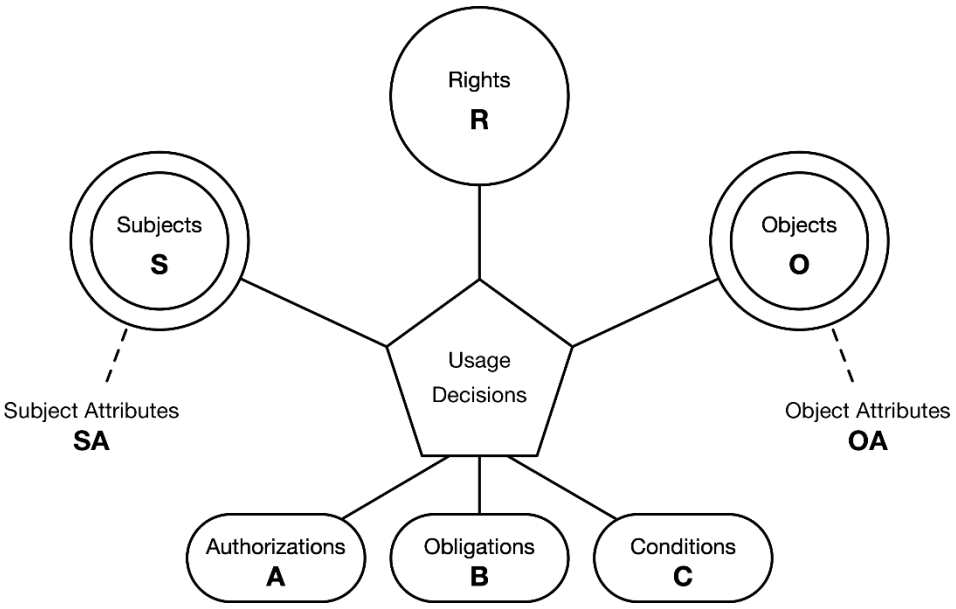


Figure 3.5 - Components of the Usage Control Model

In the following, we recall the UCON core components, shown in Figure 3.5.

Subjects and Objects: the entities who exercise their rights on the objects by performing actions on them.

Actions: the operations performed by the subject on the resource.

Attributes: information elements paired to subjects, objects, actions, and environment to describe their features. An attribute is immutable when its value can be updated only through an administrative action. An example of immutable attribute is the subject's role in systems using the Rule Based Access Control (RBAC) paradigm, as updated by the system administrator, for instance as a consequence of a career advancement.

Instead, an attribute is mutable when its value changes over time because of the normal operation of the system. Some mutable attributes change their values as a consequence of the policy enforcement, because the policy includes attribute update statements that can be executed before (pre-update), during (on-update), or after (post-update) the execution of the action. As an example, let us consider the mutable attribute which represents the number of running VMs deployed by a subject on a Cloud IaaS service. An example of mutable attribute concerning the actions is the number of instances of the same action that are currently in execution.

Mutable attributes can change their values also because of other actions performed by the subjects which are not regulated by the usage control policy. For instance, the attribute that describes the physical location of the subject changes when the subject moves from one place to another. Some mutable attributes change their values due to both reasons above. For example, the balance of the subject's e-wallet could decrease because the Usage Control policy includes a pre-update statement which states that the subject must pay for executing an action, while it could increase when the subject deposits some money in their e-wallet through a bank transaction. Finally, other attributes change their values independently of the user's behavior. For instance, date, time, and CPU load are attributes of the environment which belong to this last set.

Authorizations: They are predicates that evaluate subject and object attributes and the requested right to decide whether the subject is allowed to access the object. The evaluation of the authorization predicates can be performed before executing the access (pre-authorizations), or continuously while the access is in progress (on-authorizations) in order to promptly react to mutable attribute changes.

Conditions: They are environmental or system-oriented decision factors, i.e., dynamic factors that do not depend on subjects or objects. Hence, the evaluation of conditions involves attributes of the environment and of an action, and it can be executed before (pre-conditions) or during (on-conditions) the execution of the action.

Obligations: They are decision factors considered for verifying whether a subject has satisfied some mandatory requirements before performing an action (pre-obligations), or whether a subject continuously satisfies these requirements while performing the access (on-obligations). Obligations can be enforced after the execution of the action as well (post-obligations), but in this case they cannot affect the execution of the action.

Continuity of Policy Enforcement: Attribute mutability introduces the need to continuously execute the Usage Decision process, i.e., while an access is in progress. This is because the values of the attributes that have previously authorized the access could change in such a way that the access right does not hold any longer. In this case, the access is revoked as soon as the policy violation is detected.

The Usage Control model can be successfully adopted in case of long-standing accesses because the decision process consists of two phases: i) the pre-decision phase corresponds to traditional access control, where the decision process is performed at request time, to produce an access decision; ii) the on-decision phase is executed after the access has started and implements continuity of control, as a specific feature of the UCON model. Continuous control implies that policies are re-evaluated each time mutable attributes change their values. The

pre-decision process evaluates pre-authorizations, pre-conditions as well as pre-obligations, and access is not permitted if a policy violation is detected. The on-decision process continuously evaluates on-authorizations, on-conditions and on-obligations. In this case, if a policy violation is detected, the related ongoing access is interrupted. For further details about the Usage Control model and its application to several scenarios please refer to [LAZ10].

4 Part 1 – Secure and Robust (Group) Communication for the IoT

This section presents the developed security solutions within the area "Secure and Robust (Group) Communication for the IoT".

4.1 *Group OSCORE*

The solutions and methods presented in this section pertain to the following requirements defined in [D1.2]:

- *Non-Functional Requirements: PE-28, PE-29, P-30, P-31, P-32*
- *Security Requirements: SE-30, SE-32, SE-33, SE-34, SE-35*

The solutions and methods presented in this section pertain to the following components defined in [D1.4]:

- *The "Secure Message Exchange Manager" component of the "Secure Communication Layer" module.*
- *The "Content Distribution Manager" component of the "Secure Communication Layer" module.*

Several deployments relying on CoAP group communication (see Section 3.2) also require security to be enforced. That is, the same security requirements of one-to-one communication scenarios have to be fulfilled, also in a group communication setting. In particular, end-to-end security between a message originator and the intended message recipient(s) should be achieved.

The currently available standard solutions do not provide this kind of security. In particular, as discussed in Section 3.1, the original CoAP specification suggests only DTLS as the security protocol to use. However, in group communication scenarios this results in the following issues.

First, as discussed in Section 3.6, DTLS protects communication hop-by-hop at the transport layer. Hence, it does not provide end-to-end security, which has led to the development of the OSCORE security protocol (see Section 3.7). Second, and most important, DTLS does not support group communication, e.g., over IP Multicast. As a consequence, there is currently no standard solution to secure group communication, and especially end-to-end.

In order to fill this gap, the current standard proposal Group Object Security for Constrained RESTful Environments (Group OSCORE) [TIL23i] extends and adapts OSCORE [SEL19] to work also in group communication scenarios. In particular, Group OSCORE provides end-to-end security of CoAP messages exchanged between members of a group, e.g., using IP multicast.

Specifically, Group OSCORE ensures cryptographic binding between a CoAP group request, sent by a client to multiple servers, and the corresponding CoAP responses individually sent by the servers in the group.

However, group communication for CoAP generously admits different types of "long exchanges" for a client within a group. That is, if the client sends a request, the client may receive multiple responses from the same server to that request. Unlike in one-to-one CoAP communication, this is not limited to the case where the request is specifically an Observe request [HAR15]. Instead, a server in the group can reply with multiple responses to the same request, if: i) the request is not an Observe request altogether, but it is a group request sent to multiple recipients; or ii) the request is an Observe request (sent to one or multiple recipients), but that server does not

confirm the observation and follows-up with regular, non-notification responses.

In either case, until the pre-defined lifetime of the CoAP token associated with the group request expires or the ongoing Observation is terminated, a client has to correctly handle the reception of multiple responses from the same server to a group request. Obviously, this also applies to the secure processing of such responses.

To this end, Group OSCORE ensures that a client correctly performs anti-replay checks on such multiple responses, and that the client correctly enables its application to order the responses from a server with respect to one another. This relies on the same, original approach used for Observe notification responses protected with OSCORE, now also applied to any response to group requests and protected with Group OSCORE.

Furthermore, since message protection builds on commonly shared, group keying material, source authentication of messages exchanged in the group is achieved by using asymmetric keying material. This consists in using either digital signatures (see Section 4.1.4) or symmetric pairwise keys derived from asymmetric, individual keying material (see Section 4.1.5).

To this end, Group OSCORE defines two different modes of operation, as different ways to protect CoAP messages. The application can decide in which particular mode a particular message has to be protected.

- In the *group mode*, Group OSCORE requests and responses are encrypted with symmetric keying material as well as digitally signed, by using the private key of the sender CoAP endpoint. The group mode also supports signature verification by intermediaries external to the OSCORE group, e.g., gateways.
- In the *pairwise mode*, two group members can exchange unicast requests and responses, as protected only with symmetric keys and not including a signature. These symmetric keys are derived from Diffie-Hellman shared secrets, calculated with the asymmetric keys of the two group members. As a signature is not included, this results in a smaller message overhead. This method is intended for one-to-one messages sent in the group, i.e., it is applicable to all responses, as individually sent by servers, and to requests addressed to an individual server.

Just as OSCORE, Group OSCORE provides message binding of responses to requests, which in turn provides ordering of responses, as well as replay protection of requests. In particular, Group OSCORE fulfils the following security objectives:

- data replay protection;
- source authentication;
- message integrity;
- group-level data confidentiality (in group mode) or pairwise data confidentiality (in pairwise mode);
- proof of group membership, i.e., a message recipient is able to assert whether the message sender is a current group member;
- group privacy, i.e., an adversary cannot track a user across two OSCORE groups, unless the adversary is also a member of both such groups.

In one-to-one communication setups, it is possible for CoAP clients to discover the link to CoAP resources at a target server, typically by accessing the /.well-known/core resource hosted by that server, or via other means such as a Resource Directory [AMS22]. As a response, a CoAP client obtains a list of links to the target server's resources. In turn, any of such links can specify the target attribute "osc", thus indicating that communications must be protected with OSCORE in order to access the resource at the server targeted by that link.

In group communication setups, the same discovery approach can be used by a CoAP client, also in order to discover at once a set of group resources hosted by CoAP servers in a group. Similarly, and in order to signal the use of Group OSCORE, the target attribute "gosc" has been defined. That is, links obtained in reply to discovery lookup requests (from /.well-known/core at a target server or via other means) can specify the target attribute

"gosc", which indicates that communications must be protected with Group OSCORE in order to access the resources targeted by such links. To ensure compatibility with devices not supporting Group OSCORE, a resource link specifying the target attribute "gosc" must also specify the target attribute "osc".

Finally, just like OSCORE, Group OSCORE is independent of the specific transport layer, and it works wherever CoAP works. Also, like with OSCORE, it is possible to combine Group OSCORE with communication security on other layers. One example is the use of transport layer security, such as DTLS [RES12], between one client and one proxy (and vice versa), or between one proxy and one server (and vice versa), to protect the routing information of packets from observers. Note that, as discussed above, DTLS cannot be used to secure messages sent over IP multicast.

A Java implementation of Group OSCORE from RISE is available at [GOSC-DEVa], with plans for integration in the Californium library [CALIFORNIUM] from the Eclipse Foundation, as available for use in the SIFIS-Home project. Also, it has been individually demonstrated through the early focused demo described in Appendix C, and then integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 and the Smart-Home pilot within WP6.

Furthermore, an ongoing Contiki-NG implementation of Group OSCORE from RISE is available at [GOSC-DEVb], as intended to be integrated in the Contiki-NG operating system [Contiki-NG]. An early experimental performance evaluation of Group OSCORE has been performed and published in [GUN22], based on the implementation above and involving real resource-constrained IoT devices.

The rest of this section is organized as follows. Section 4.1.1 introduces the Group Manager, i.e., an entity required to operate and maintain an OSCORE group. Section 4.1.2 describes the main differences and extensions of Group OSCORE compared to OSCORE. Section 4.1.3 discusses the main points and implications of distributing new keying material in the OSCORE group. Section 4.1.4 describes the message protection/verification of Group OSCORE when using the group mode, in comparison with OSCORE taken as baseline. Section 4.1.5 describes the pairwise mode of Group OSCORE. Section 4.1.6 shortly mentions ongoing related work pertaining to secure group communication for CoAP. Finally, Section 4.1.7 provides a high-level overview of the experimental evaluation of Group OSCORE published in [GUN22].

4.1.1 The OSCORE Group Manager

Group OSCORE relies on the presence of an additional Group Manager entity. This is responsible for one or more OSCORE groups, for the respective Group Identifier (Gid) used as OSCORE ID Context, and for the Sender ID and Recipient ID of the respective group members.

The Group Manager has exclusive control of the Gid values uniquely assigned to the different groups under its control, and of the Sender ID and Recipient ID values uniquely assigned to the members of each of those groups. A CoAP endpoint receives the Gid and other OSCORE input parameters, including its own Sender ID, from the Group Manager upon joining the OSCORE group. That Sender ID is valid only within that group, and is unique within the group.

Furthermore, the Group Manager stores and maintains the public authentication credentials of endpoints joining a group (as including the public keys of those endpoints), and provides information about the group and its members to other current group members. Then, a group member can retrieve from the Group Manager the public authentication credential and other information associated with other group members.

Finally, it is recommended that the Group Manager takes care of the group joining by using the approach described in Section 6.1 and defined in [TIL2b], as based on the ACE framework for authentication and authorization in constrained environments [SEI22] (see Section 3.8).

4.1.2 Main Differences From OSCORE

This section introduces in what respects Group OSCORE mainly differs from OSCORE [SEL19], with a focus on the data structure and keying material stored by group members, as well as the COSE object and compressed encoding of OSCORE messages.

As a particular case, a group member can assume the special role of "silent server". This kind of endpoint is interested in receiving request messages, but never replies to them. For example, a simple lighting device can be configured to never send responses if the user has visual access to the physical environment, with further advantages in terms of network latency and reliability. Also, a device can act as a network monitor, thus silently listening to and logging messages exchanged in the group, while never replying to requests sent by other group members. An endpoint can implement both a silent server and a client, as the two roles are independent. However, an endpoint implementing only a silent server processes only incoming requests, and, in case it uses only the group mode, it maintains less keying material and especially does not have a Sender Context for the OSCORE group.

A description of the actual message processing is provided in Section 4.1.4 for the group mode and in Section 4.1.5 for the pairwise mode.

Each CoAP endpoint as member of an OSCORE group stores a Security Context (see Section 3.7.1). Compared to the original format used in OSCORE, the Security Context is extended as follows and as shown in Table 4.1. Further details are provided in Sections 4.1.2.1 and 4.1.2.2. The elements marked with (*) are relevant only if the group mode is used in the group. The elements marked with (^) are relevant only if the pairwise mode is used. If an element is not relevant, it remains unset in the Security Context.

- One Common Context, shared by all the endpoints in the OSCORE group. The following new parameters are included in the Common Context: the public authentication credential of the Group Manager (including the Group Manager's public key); the Group Encryption Algorithm; the Signature Algorithm; the Signature Encryption Key; and the Pairwise Key Agreement Algorithm.
- One Sender Context, extended with the endpoint's private key and the Pairwise Sender Keys to use with each other endpoint. The Sender Context is omitted if the endpoint is configured exclusively as silent server and uses only the group mode.
- One Recipient Context for each endpoint from which messages are received. No Recipient Contexts are maintained as associated with endpoints from which messages are not (expected to be) received. The Recipient Context is extended with the public authentication credential of the associated endpoint (including the other endpoint's public key) as well as with the Pairwise Recipient Key to use with that endpoint.

Context component	**New information element**
Common Context	Authentication Credential Format Group Manager Authentication Credential * Group Encryption Algorithm * Signature Algorithm * Signature Encryption Key ^ Pairwise Key Agreement Algorithm
Sender Context	Endpoint's own private key Endpoint's own authentication credential ^ Pairwise Sender Keys for the other Endpoints
Each Recipient Context	Public authentication credential of the other endpoint ^ Pairwise Recipient Key for the other Endpoint

Table 4.1 - Additions to the OSCORE Security Context.

4.1.2.1 Common Context

The following clarifies the content of the Common Context, as deltas and additions from what is defined for OSCORE [SEL19] (see Section 3.7).

The AEAD Algorithm parameter is inherited from the Common Context defined in OSCORE, and specifies the AEAD algorithm used for decryption and decryption of messages protected with the pairwise mode. This parameter is not set when the pairwise mode is not used in the group.

The ID Context parameter is inherited from the Common Context defined in OSCORE, and specifies the Group Identifier (Gid) of the OSCORE group, which is thus used as ID Context for that group. The choice of the Gid is specific to the application running at the Group Manager. It is up to the application running at the group members how to handle possible collisions between Gids, as used for OSCORE groups managed by different, non-synchronized Group Managers.

The Common IV parameter is inherited from the Common Context defined in OSCORE, and specifies the Common IV to use for building the AEAD nonces used for message encryption and decryption. Unlike in OSCORE, the length of the Common IV is determined as follows. If only one among the AEAD Algorithm and the Group Encryption Algorithm is set, the length of the Common IV is the AEAD nonce length for the set algorithm. Instead, if both the AEAD Algorithm and the Group Encryption Algorithm are set, the length of the Common IV is the greatest AEAD nonce length among those of the two algorithms. This ensures that only one Common IV is stored, as derived according to the same derivation process of OSCORE (see Section 3.2 of [SEL19]) from a set of input parameters (see Section 3.2 of [SEL19]). When performing encryption/decryption operation on outgoing/incoming messages, the right number of bytes is taken from the stored Common IV as actual Common IV to use, depending of the AEAD nonce size of the specifically used AEAD algorithm for the mode of operation used to process the message in question.

The Authentication Credential Format parameter specifies the format of authentication credentials in the group.

The public authentication credential of the Group Manager must be provided to the recipient endpoint together with a proof-of-possession of the corresponding private key, for instance when the recipient endpoint joins the OSCORE group. The public key of the Group Manager is used as part of the Additional Authenticated Data (AAD) when protecting and unprotecting a message (see Section 4.1.2.3).

The Group Encryption Algorithm parameter identifies the encryption algorithm used to protect messages when using the group mode of Group OSCORE (see Section 4.1.4). Signature Algorithm identifies the digital signature algorithm used to compute a counter signature on the COSE object when using the group mode. The Counter Signature Algorithm has to be selected among the signing ones available in COSE (see https://www.iana.org/assignments/cose/cose.xhtml#algorithms). These parameters are not set when the group mode is not used in the group.

The Pairwise Key Agreement Algorithm parameter specifies the algorithm used to derive pairwise symmetric keys, when using the pairwise mode of Group OSCORE (see Section 4.1.5). This parameter is not set when the pairwise mode is not used in the group.

The parameters associated with the Signature Algorithm and the Pairwise Key Agreement Algorithm are embedded in the stored public authentication credentials of the Group Manager and of the group members.

The Signature Encryption Key is common to all the group members, is derived by means of the same key derivation process of OSCORE (see Section 3.2 of [SEL19]), and is used to further and separately encrypt the signature of messages protected with the group mode of Group OSCORE.

4.1.2.2 Sender Context and Recipient Context

Group OSCORE uses the same derivation process of OSCORE (see Section 3.2 of [SEL19]) to derive Sender Context and Recipient Context – and specifically Sender/Recipient Keys – from a set of input parameters (see Section 3.2 of [SEL19]). However, in Group OSCORE, the Sender Context and Recipient Context additionally contain asymmetric keys.

In particular, the Sender Context includes the private key of the endpoint. When using the group mode (see Section 4.1.4), the private key is used to generate a signature included in the sent Group OSCORE message. When using the pairwise mode (see Section 4.1.5), the private key is used to derive a pairwise key between the endpoint and another member of the OSCORE group. It is out of scope for Group OSCORE how the private key has been established.

Each Recipient Context includes the public authentication credential of the associated other endpoint. The public key specified in the authentication credential is used to verify the signature of a Group OSCORE message received from that other endpoint when using the group mode (see Section 4.1.4), or to derive a pairwise key for verifying Group OSCORE messages from that other endpoint protected with the pairwise mode (see Section 4.1.5).

The public authentication credential of the associated endpoint as well as the input parameters for deriving the Recipient Context parameters may be provided to the recipient endpoint upon joining the OSCORE group. Alternatively, these parameters can be acquired at a later time, for example the first time a message is received from this particular other endpoint in the OSCORE group. The received message, together with the Common Context, includes everything necessary to derive a Recipient Context for verifying a message, except for the public authentication credential of the associated other endpoint.

For particularly constrained devices, it can be not feasible to simultaneously handle the ongoing processing of a recently received message and the retrieval of the associated endpoint's public authentication credential. Such devices may instead be configured to drop a received message for which there is currently no Recipient Context, and retrieve the public authentication credential of the sender endpoint in order to have it available to verify subsequent messages from that endpoint.

4.1.2.3 COSE Object

Compared to OSCORE (see Section 3.7.2), the following differences apply to the COSE object.

- When using the group mode (see Section 4.1.4), the COSE Object includes an additional signature. Its value is set to the counter signature of the Encrypted COSE object, computed by the sender endpoint as described in [SCH22c], by using its own private key and according to the Signature Algorithm specified in the Security Context. The signature is computed over the AAD and the ciphertext of the encrypted COSE object.
- The 'kid' parameter is present in all messages, i.e., both requests and responses, specifying the Sender ID of the endpoint transmitting the message. An exception is possible only for response messages, if sent as a reply to a request protected in pairwise mode.
- The 'kid context' parameter is present in every request message, specifying the Group Identifier value (Gid) of the group's Security Context. This parameter remains optional to include in response messages.
- The AAD takes an extended format than the one used in OSCORE [SEL19], in order to include the following additional information: the algorithms specified in the Common Context; the Gid used when protecting a request message, i.e., the new ‘request_kid_context’ element; the binary serialization of the OSCORE Option; the public authentication credential of the sender endpoint and the public authentication credential of the Group Manager. Note that, like in OSCORE, the AAD is not transmitted, but only takes part in computational operations during the message encryption/decryption.

4.1.2.4 Compressed Message Encoding

Compared to OSCORE (see Section 3.7.2), the following differences apply to the encoding of an OSCORE message.

- When using the group mode (see Section 4.1.4), the ciphertext of the COSE object as payload of the OSCORE message is further concatenated with the value of the countersignature of the encrypted COSE object (see Section 4.1.2.3). After that, the countersignature is separately further encrypted through a keystream derived from the Signature Encryption Key (see Section 4.1.4).
- The sixth least significant bit, namely the Group Flag bit, in the first byte of the OSCORE option containing the OSCORE flag bits is used to signal the usage of the group mode (see Section 4.1.4). In particular, the Group Flag bit is set to 1 if the OSCORE message is protected using the group mode. In any other case, including when using the pairwise mode (see Section 4.1.5), this bit is set to 0.

4.1.3 Renewal of Group Keying Material

Due to a number of reasons, the Group Manager may force the members of an OSCORE group to establish a new Security Context, by revoking the current group keying material and distributing new one (rekeying).

To this end, a new Group Identifier (Gid) for the OSCORE group and a new value for the Master Secret parameter is distributed to the group members. When doing so, the Group Manager may also distribute a new value for the Master Salt parameter, while it should preserve the same current value of the Sender ID of each group member.

After that, each group member re-derives the keying material in its own Sender Context and Recipient Contexts, as described in Section 4.1.2, by using the newly distributed Gid and Master Secret parameter. The Master Salt used for the re-derivations is the newly distributed Master Salt parameter if provided by the Group Manager, or an empty byte string otherwise. Thereafter, each group member must use its latest installed Sender Context to protect its own outgoing messages.

Note that the distribution of a new Gid and Master Secret parameter may result in group members temporarily storing misaligned Security Contexts. Specifically, a group member may become not able to process messages received right after the distribution of a new Gid and Master Secret parameter.

Every time a current endpoint leaves the group, the Group Manager renews the group keying material and informs the remaining members about the leaving endpoint. This preserves the capability of group members to correctly assert the group membership of a message sender, and additionally preserves forward security in the group. Depending on the specific application requirements, it is recommended to rekey the group also every time a new joining endpoint is added to the group, thus preserving also backward security.

Group OSCORE is not devoted to a particular method or key management scheme for rekeying the OSCORE group. However, it is recommended that the Group Manager supports the distribution of the new Gid and Master Secret parameter to the OSCORE group according to the Group Rekeying Process described in Section 6.1 and defined in [TIL23a].

4.1.4 Group Mode

This section describes how Group OSCORE protects messages in group mode, as a sequence of deltas compared to the message processing of OSCORE [SEL19] (see Section 3.7.2).

In particular, source authentication of messages is achieved by appending a signature to the message payload, computed by using the private key of the message sender. On the other end, message confidentiality is achieved at a group level, i.e., every other member of the OSCORE group is able to decrypt a message protected in group mode.

A client protects a request in group mode as in OSCORE, with the following differences.

- The extended Additional Authenticated Data (AAD) defined in Section 4.1.2.3 is used for encrypting and signing the request.

- The encryption and the encoding of the COSE object are as defined in Sections 4.1.2.3 and 4.1.2.4, respectively. In particular, the Group Flag bit is set to 1.
- A countersignature of the Encrypted COSE Object is also computed and added at the end of the payload of the protected request message. After that, the countersignature is separately, further encrypted. The encrypted countersignature is computed by *xoring* the plain countersignature with a keystream derived from the Signature Encryption Key in the Common Context.
- For every "long exchange", the client has to store the values that the Gid and its own Sender ID have when the exchange starts, used as 'kid context' and 'kid' parameter in the original request. The client must not update those stored values, even in case it receives a new Sender ID from the Group Manager or the whole group is rekeyed. This makes it possible to preserve an ongoing long exchange even across a group rekeying (see Section 4.1.3).

A server verifies a request in group mode as in OSCORE, with the following differences.

- The decoding of the compressed COSE object follows the updates in Section 4.1.2.4.
- If the server discards the request due to not retrieving a Security Context associated with the OSCORE group, the server may respond with a 4.02 (Bad Option) error.
- If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Gid ('kid context'), then the server may create a new Recipient Context and initialize it at that point in time, also retrieving the client's public authentication credential. Such a configuration is application specific. If the application does not specify dynamic derivation of new Recipient Contexts, the server stops processing the request.
- The extended Additional Authenticated Data (AAD) defined in Section 4.1.2.3 is used, for decrypting the request and verifying the countersignature of the encrypted COSE object.
- The server decrypts the received countersignature by xoring it with the same keystream used by the client and derived from the Signature Encryption Key. Then, before decrypting the request, the server also verifies the recovered plain countersignature using the public key of the client from the associated Recipient Context. If the signature verification fails, the server may reply with a 4.00 (Bad Request) response.
- If the used Recipient Context was created upon receiving this group request and the message is not decrypted and verified successfully, the server may delete that Recipient Context. Such a configuration, which is application specific, prevents attacks aimed at overloading the server's storage and creating processing overhead on the server.
- For each newly started long exchange, the server stores the values of the 'kid context' and 'kid' parameters from the original request, i.e., the Gid and the Sender ID of the client at that time. Then, the server does not update those stored values, even if the client gets and starts using a new Sender ID received from the Group Manager, or the whole group is rekeyed (see Section 4.1.3).

A server protects a response in group mode as in OSCORE, with the following differences.

- The encoding of the compressed COSE object follows the updates in Section 4.1.2.4. In particular, the Group Flag bit is set to 1.
- With the possible exception of the first response in the long exchange, the server must encode the Partial IV in the protected response, as set to its own Sender Sequence Number value, and use that Partial IV to build the AEAD nonce for the encryption process. Then, the server must increment the Sender Sequence Number by one, and must include in the response the 'Partial IV' parameter, which is set to the Partial IV above.
- The extended AAD defined in Section 4.1.2.3 is used, for encrypting and signing the response.
- A countersignature of the encrypted COSE object is also computed and added at the end of the payload of the protected response message. After that, the countersignature is separately, further encrypted. The encrypted countersignature is computed by xoring the plain countersignature with a keystream derived from the Signature Encryption Key.

- The server may have ongoing long exchanges with the client, started by requests protected with an old Security Context. Then, the following applies.
    - After completing the establishment of a new Security Context, e.g., upon group rekeying, the server must protect the following responses with its own Sender Context from that new Security Context.
    - For each ongoing long exchange, the server should include in the first response protected with the new Security Context also the 'kid context' parameter, which has its value set to the ID Context of the new Security Context, i.e., the new Group Identifier (Gid). The server can optionally include the 'kid context' parameter, as set to the new Gid, also in the further following responses for those long exchanges.
    - For each ongoing long exchange, the server has to use the stored values of the 'kid context' and 'kid' parameters from the original request, i.e., the Gid and the Sender ID of the client at the time of the original request, as value for the 'request_kid_context' and 'request_kid' elements in the AAD (see Section 4.1.2.3), when protecting responses for that long exchange.

Since a group rekeying can occur, with consequent re-establishment of the Security Context, the server must always protect a response by using its own Sender Context from the latest owned Security Context. As a consequence, right after a group rekeying has been completed, the server may end up protecting a response by using a Security Context different from the one used to protect the request. In such a case, the server proceeds as follows.

- The server must encode the Partial IV in the protected response, as set to its own Sender Sequence Number value, and use that Partial IV to build the AEAD nonce for the encryption process. Then, the server, must increment the Sender Sequence Number by one, and must include in the response the 'Partial IV' parameter, which is set to the Partial IV above.
- The server should include in the response the 'kid context' parameter, which is set to the ID Context of the new Security Context, i.e., the new Group Identifier (Gid).

A client verifies a response in group mode as in OSCORE, with the following differences.

- The decoding of the compressed COSE object follows the updates in Section 4.1.2.4.
- The extended AAD defined in Section 4.1.2.3 is used, for decrypting the response and verifying its counter signature.
- If the request was protected in pairwise mode, the following applies.
    - The client checks that the replying server is the expected one, by relying on the server's public authentication credential used to verify the countersignature of the response and earlier used to derive the Pairwise Sender Key for encrypting the request.
    - The client assumes the Recipient ID to be the same one considered when sending the request, in case a ‘kid’ parameter is not included in the received response.
- If the Recipient ID ('kid' of the response) does not match with any Recipient Context for the retrieved Gid ('kid context'), then the client may create a new Recipient Context and initialize it at that point in time, also retrieving the server’s public authentication credential. Such a configuration is application specific. If the application does not specify dynamic derivation of new Recipient Contexts, the client stops processing the response.
- The client decrypts the received countersignature by *xoring* it with the same keystream used by the server and derived from the Signature Encryption Key. Then, before decrypting the response, the client also verifies the recovered plain countersignature using the public key of the server from the associated Recipient Context. If the signature verification fails, the client stops processing the response.
- If the used Recipient Context was created upon receiving this response and the message is not decrypted and verified successfully, the client may delete that Recipient Context. Such a configuration, which is application specific, prevents attacks aimed at overloading the client's storage and creating processing overhead on the client.

As discussed above, a client may receive a response protected with a Security Context different from the one

used to protect the corresponding group request.

For each ongoing long exchange, the client has to use the stored values of the 'kid context' and 'kid' parameters from the original request that started the exchange, i.e., the Gid and its own Sender ID at the time of the original request, as value for the 'request_kid_context' and 'request_kid' elements in the AAD (see Section 4.1.2.3), when decrypting responses for that exchange and verifying their signatures. This ensures that, during the exchange's lifetime and across a group rekeying (see Section 4.1.3), the client is able to correctly verify responses within the exchange, even if it is individually rekeyed and starts using a new Sender ID received from the Group Manager or the whole group is rekeyed.

4.1.5 Pairwise Mode

The pairwise mode of Group OSCORE is intended to support one-to-one message exchanges among group members. In particular, the pairwise mode protects a message by using only symmetric keys, as derived by using the public/private keys of the two communicating endpoints. Besides, the pairwise mode does not include any digital signature in the protected message, while still ensuring source authentication. A sender endpoint must not use the pairwise mode to protect a message intended to multiple recipients or to the whole group, e.g., sent over IP multicast.

Especially for a CoAP request addressed to an individual server in the group, the pairwise mode should be used, rather than the group mode. This ensures that the request is indeed received and decrypted only by the exact server intended as recipient.

Otherwise, since Group OSCORE (just as OSCORE and DTLS) does not protect addressing information at the lower layers, an active adversary would be able to intercept a unicast request protected in group mode, and redirect it to a different server in the group than the intended one. Such a server would still successfully verify the request, which can have severe consequences especially in case of unsafe REST methods, i.e., POST, PUT, PATCH and DELETE. In fact, it is not recommended for a client to protect a unicast request message by using the group mode, in order to prevent such attacks altogether.

Relevant cases where a client simply has to send unicast requests to a particular server in the group include, but are not limited to: i) the exchange of messages including a CoAP Echo [AMS20] option, to prove reachability and message freshness on the server side; and ii) the execution of Blockwise [BOR16] transfer operations as limited to happen over unicast.

The usage of the pairwise mode has the following limitations:

- It is not usable in use cases that include intermediaries as signature verifiers external to the OSCORE group. These include, for instance, gateways deployed to forward CoAP messages, upon successful verification of an outer countersignature. While this is indeed possible to do with messages protected in group mode, the pairwise mode would prevent such gateways to perform their task.
- It requires endpoints to support signature algorithms that support both a signature and encryption scheme. These include, for instance, ECDSA and EdDSA. In particular, ECDSA can be used “as is” both for signature operations as well as derivation of symmetric shared secrets. Instead, EdDSA relies on using a particular elliptic curve (e.g., the Ed25519 Edward curve) for signature operations, and requires a remapping to different curve coordinates (e.g., the X25519 Montgomery curve) to perform the derivation of symmetric shared secrets. On the contrary, these algorithms do not include RSA, that can be used only for signature operations.

The pairwise mode relies on an additional key derivation process, which is described in Section 4.1.5.1. Then, Section 4.1.5.2 describes the message processing performed in pairwise mode.

4.1.5.1 Pairwise Key Derivation

If they support the pairwise mode of Group OSCORE, two members in an OSCORE group can derive symmetric

pairwise keys, by using two main inputs: i) their own Sender/Recipient Key; ii) a static-static Diffie-Hellman shared secret [BAR18]. Then, a pairwise key is used to protect a message, using the same AEAD Algorithm specified in the Common Context.

The actual key derivation process is described below, and relies on the same construction used in OSCORE [SEL19] for establishing the Security Context.

Pairwise Sender Key = HKDF(*Sender Key, IKM-Sender, info, L*)
Pairwise Recipient Key = HKDF(*Recipient Key, IKM-Recipient, info, L*)

with
IKM-Sender = *Sender Auth Cred* | *Recipient Auth Cred* | *Shared Secret*
IKM-Recipient = *Recipient Auth Cred* | *Sender Auth Cred* | *Shared Secret*

where:

- The Pairwise Sender Key is the AEAD key for processing outgoing messages addressed to endpoint X.
- The Pairwise Recipient Key is the AEAD key for processing incoming messages from endpoint X.
- HKDF is the OSCORE HKDF algorithm from the Common Context.
- The Sender Key from the Sender Context is used as salt in the HKDF, when deriving the Pairwise Sender Key.
- The Recipient Key from the Recipient Context associated with endpoint X is used as salt in the HKDF, when deriving the Pairwise Recipient Key.
- Sender Auth Cred is the endpoint's own authentication credential from the Sender Context.
- Recipient Auth Cred is the endpoint X's authentication credential from the Recipient Context associated with the endpoint X.
- The Shared Secret is computed as a cofactor Diffie-Hellman shared secret (see Section 5.7.1.2 of [BAR18]), using the Pairwise Key Agreement Algorithm specified in the Common Context. The endpoint uses its private key from the Sender Context and the Recipient Public Key. For curves X25519 and X448, the procedure is described in Section 5 of [LAN16], possibly using signing public keys first mapped to Montgomery coordinates.
- IKM-Sender is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Sender Key. IKM-Sender is the byte string concatenation of the Sender Public Key, the Recipient Public Key, and the Shared Secret.
- IKM-Recipient is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Recipient Key. IKM-Recipient is the byte string concatenation of the Recipient Public Key, the Sender Public Key, and the Shared Secret.
- The 'info' and 'L' parameters are as defined for the establishment of the Security Context of OSCORE (see Section 3.2.1 of [SEL19]). That is: the 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context; L and the 'L' element of the 'info' array are the size of the key for the AEAD Algorithm from the Common Context, in bytes.

The security of using the same key pair for Diffie-Hellman and for signing is proven in [DEG11] and [THO21]. In particular, [THO21] is the first contribution that provides a formal, cryptographic proof of the following claim: an asymmetric key pair intended for digital signing operations with the Edward elliptic curves Ed25519 and Ed448 or with the NIST elliptic curves (e.g., P-256) can securely be used also to enable key agreement operations based on Diffie-Hellman. In case Edward curves are used, this relies on a preliminary conversion of coordinates from the curve Ed25519 (Ed448) to the curve X25519 (X448). The proof especially considers the double purpose that asymmetric key pairs have in the security protocol Group OSCORE, where such a property allows every communicating peer to rely on a single key pair for both signing and key agreement operations, hence reducing the storage overhead and simplifying related key management operations.

Therefore, it is indeed secure for every member of an OSCORE group to rely on a single asymmetric key pair for itself, and on a single public key for each other group member. In particular, this is sufficient to securely enable

both signature operations performed in the group mode, as well as key agreement operations to derive pairwise keys used in the pairwise mode. As a result, this also reduces the storage overhead and simplifies related key management operations in the OSCORE group.

Practically, if ECDSA asymmetric keys are used (e.g., with curve P-256), they can be securely used as is also for Diffie-Hellman key agreement operations, in addition to signing operations. Instead, if EdDSA asymmetric keys are used, the Edward coordinates from curve Ed25519 (Ed448) have to be re-mapped into Montgomery coordinates, by using the maps defined in [LAN16], before using the X25519 (X448) function from [LAN16].

When using any of its pairwise keys, a sender endpoint including the 'Partial IV' parameter in the protected message has to use the current, fresh value of its own Sender Sequence Number, from its own Sender Context (see Section 4.1.2). In fact, at each endpoint, the same Sender Sequence Number space is used for all outgoing messages sent by that endpoint to the group and protected with Group OSCORE. This has the benefit to limit both storage and complexity.

Once completed the establishment of a new Security Context, e.g., following a group rekeying (see Section 4.1.3) or the assignment of a new Sender ID from the Group Manager, a group member must delete all the pairwise keys it stores. In fact, as new Sender/Recipient keys have been derived, those must be used to possibly derive new pairwise keys. On the other hand, as long as any two group members preserve the same asymmetric keys, the Diffie-Hellman shared secret does not change across updates of the group keying material.

4.1.5.2 Message Processing

To protect a message in pairwise mode, a sender endpoint needs to know the public authentication credential and the Recipient ID for the recipient endpoint, as stored in its own Recipient Context associated with that recipient endpoint.

Also, the sender endpoint has to know the individual, unicast address of the recipient endpoint. To make this information available and facilitate its retrieval, servers may provide a resource to which clients in the group can send a group request for obtaining addressing information, e.g., as sent over IP multicast. For instance, such a group request can aim at discovering a server identified by its 'kid' value, or a set thereof. The specified set may be empty, hence identifying all the servers in the group.

The processing of messages using the pairwise mode is very similar to the one defined for OSCORE [SEL19], from which the following differences apply.

- The 'kid' and 'kid context' parameters of the COSE object are used as per Section 4.1.2.3.
- The extended AAD defined in Section 4.1.2.3 is used for the encryption process.
- The Pairwise Sender/Recipient Keys used as Sender/Recipient keys are derived as defined in Section 4.1.5.1.

When using the pairwise mode, messages are protected and unprotected as in OSCORE [SEL19], with the differences summarized above. Also, within long exchanges in the group, the same additions defined in Section 4.1.4 for the group mode apply, as to the handling of the 'kid' and 'kid context' parameters. Furthermore, the following applies:

- Failure on the server side when processing a request may result in returning an error message.
- If the server is using a different Security Context for the response compared to what was used to verify the request, then the server must include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response.
- If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request, then the server must include the new ID Context in the 'kid context' parameter of the response.
- The server may have received a new Sender ID from the Group Manager. In such a case, the server should include the 'kid' parameter in the response even when the request was also protected in pairwise mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

- If the request was protected in pairwise mode, the following applies when the client receives a response also protected in pairwise mode.
    - The client checks that the replying server is the expected one, by relying on the server's authentication credential used to derive the Pairwise Recipient Key for decrypting the response.
    - The client assumes the Recipient ID to be the same one considered when sending the request, in case a 'kid' parameter is not included in the received response.

4.1.6 Additional Ongoing Activities

Notably, additional design and development activities have also been ongoing within the SIFIS-Home project to define additional, advanced services for secure group communication. Due to time constraints and prioritization choices, these results have still not reached a maturity level that could enable their integration in the SIFIS-Home solution. For information, those activities are shortly summarized below.

First, additional mechanisms are required to fully support bidirectional message exchanges in a group where an intermediary proxy is also deployed, between an origin client and the origin servers. Relevant issues in this respect are being addressed in the proposal [TIL23e]. Besides, this can conveniently rely on the ongoing proposal at [TIL23f] for enabling the use of OSCORE and Group OSCORE also at proxies. In the case at hand, a proxy would thus become able to authenticate and identify an origin client as allow-listed by leveraging their (Group) OSCORE secure association, before forwarding a request from that client to a group of servers. This also opens for multiple, nested protections of a same CoAP message, through multiple (Group) OSCORE protection layers applied in sequence.

Second, the "Observe" extension for CoAP [HAR15] (see Section 3.1) has been recently enabled also in groups, i.e., one client endpoint can simultaneously observe a shared group resource at multiple servers. However, some group applications (e.g., publish-subscribe) would benefit from a reversed pattern, where multiple clients observe the same resource at the same server. To this end, the ongoing proposal at [TIL23d] has been defining how a CoAP server can provide such a functionality, by sending one single notification response targeting all the observer clients at once (e.g., over IP multicast). This functionality is applicable also in case intermediary proxies are used, as well as in case group communications are protected end-to-end with Group OSCORE.

Finally, as originally specified in [SEL19], OSCORE does not make it possible to effectively cache protected responses at intermediary proxies, due to the impossibility for origin clients to produce a cache hit by means of an OSCORE-protected request. To fill this gap. the ongoing proposal at [AMS23] effectively enables the cacheability of OSCORE-protected responses, building on Group OSCORE constructions and on the new concept of "deterministic request". In applications intended for content distribution, this allows proxies to serve several clients' requests from their own cache, thus yielding less traffic and accesses at the origin servers, as well as achieving considerable performance improvements.

4.1.7 Implementation and Experimental Evaluation

An implementation of Group OSCORE in C for the Contiki-NG operating system has been made and is available at [OSC-DEV]. This implementation has been used to perform an experimental evaluation of Group OSCORE, with the aim of early investigating and assessing its performance and feasibility when run on IoT devices that are resource-constrained and battery-powered. A comparison was also carried out against unprotected CoAP group communication, unprotected CoAP one-to-one communication, and CoAP one-to-one communication protected with OSCORE. This section summarizes the carried-out evaluation and its main results, which are presented and discussed with further details in a published journal paper [GUN22].

Figure 4.1 depicts the network topology and network protocol stack considered for the experiments. In particular, one constrained CoAP client communicates with three constrained CoAP servers.

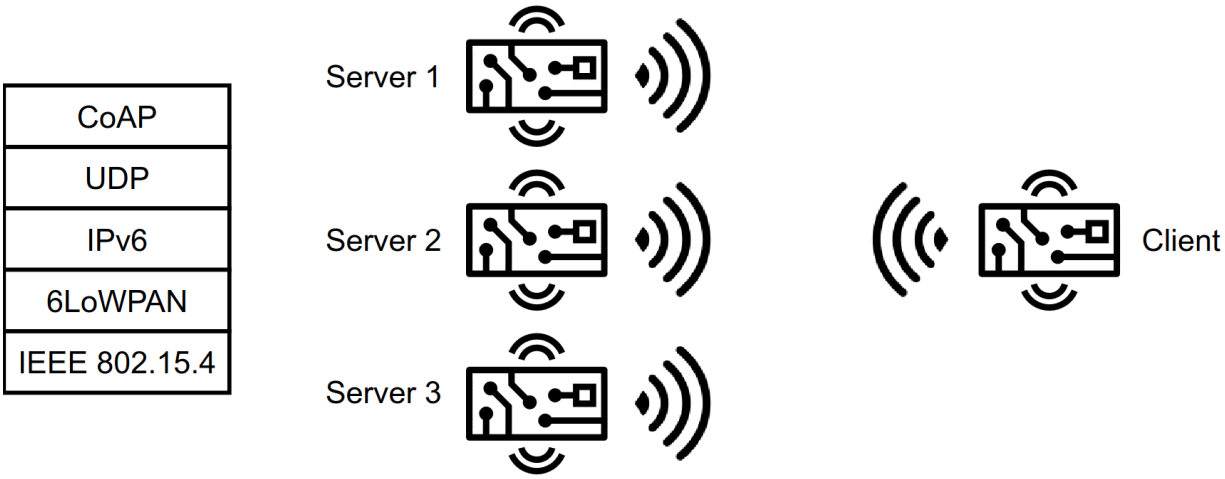


Figure 4.1 - Network topology and network stack for the evaluation experiments

Different sets of experiments were run considering two different hardware platforms for the used devices, namely Zolertia Firefly Rev. A [ZOL-FF] or TI CC1352R1 Launchpad [TI-SIM]. Table 4.2 provides an overview of the two considered hardware platforms. As embedded operating system, the devices use Contiki-NG 4.0, which relies on a network stack using 2.4 GHz IEEE 802.15.4 LR-WPAN for physical and Medium Access Control layer, with 6LoWPAN and IPv6 at the network layer, and UDP at the transport layer.

Platform	**MCU**	**Clock Freq.**	**RAM**	**ROM /Flash**	**Hardware Features**	**RF Bands**
Zolertia Firefly Rev. A (**Zoul**)	ARM Cortex M3	32MHz	32kB (16kB in low-power mode)	512kB	AES128/256, SHA2, ECC128/256	sub-GHz, 2.4 GHz
TI CC1352R1 Simplelink (**Simplelink**)	ARM Cortex M4F	48MHz	80kB + 8kB cache	352kB flash +256kB ROM	AES128/256, SHA2, ECC128/256.	sub-GHz, 2.4 GHz

Table 4.2 - Summary of the hardware platforms used in the experiments

The performed experiments considered the following four test scenarios:

1. Unprotected CoAP one-to-one communication, with no security mechanism applied, and all messages sent over unicast.
2. Unprotected CoAP group communication, with no security mechanism applied, the client sending request messages over IP multicast, and receiving corresponding response messages over unicast.
3. Protected CoAP one-to-one communication, using OSCORE to protect all messages, each of which is sent over unicast.
4. Protected CoAP group communication, using Group OSCORE to protect all messages, with the client

sending request messages over IP multicast, and receiving response messages over unicast. All messages are protected using the group mode of Group OSCORE.

Scenario (3) was in turn evaluated considering two different configurations, i.e., by performing cryptographic operations as implemented either in Software or in Hardware (as possible in both the considered platforms).

Scenario (4) was in turn evaluated considering three different configurations, by using as Signature Algorithm in the group either: i) ECDSA with curve P-256 implemented in Software, as for other cryptographic operations; of ii) or ECDSA with curve P-256 implemented in Hardware, as for other cryptographic operations; or iii) EdDSA with curve Ed25519 implemented in Software, as for other cryptographic operations. Neither of the considered platforms supports EdDSA with curve Ed25519 in hardware.

In both scenarios (3) and (4), the following applies: the software implementation of the authenticated encryption algorithm AES-CCM-128 is provided by Contiki-NG; the software implementation of the hash function SHA256 is provided by TinyDTLS [TDTLS]; the hardware implementation of AES-CCM-128, SHA256 and ECDSA P-256 for the Zoul platform is provided by Contiki-NG. The same functions for the Simplelink platform are provided by the Simplelink SDK from Texas Instruments. The software implementations of asymmetric cryptography are uECC (micro-ecc) for ECDSA with P-256 [M-ECC] and Monocypher for EdDSA with Ed25519 [MON-CY].

Thus, the following test cases were considered during the experiments, for each of the two hardware platforms.

- **COAP**: corresponding to test scenario (1) above.
- **Group-COAP**: corresponding to test scenario (2) above.
- **OSCORE-SW**: corresponding to test scenario (3) above, where cryptographic operations were implemented in Software, and hence performed on the primary CPU.
- **OSCORE-HW**: corresponding to test scenario (3) above, where cryptographic operations were implemented in hardware, and hence performed on the dedicated cryptographic accelerators of the hardware platforms.
- **Group-OSCORE-SW**: corresponding to test scenario (4) above, where the cryptographic operations were implemented in Software, and hence performed on the primary CPU. The Signature Algorithm used in the group is ECDSA with curve P-256.
- **Group-OSCORE-HW**: corresponding to test scenario (4) above, where the cryptographic operations were implemented in Hardware, and hence performed on the dedicated cryptographic accelerators of the hardware platforms. The Signature Algorithm used in the group is ECDSA with curve P-256.
- **Group-OSCORE-EDDSA**: corresponding to test scenario (4) above, where the cryptographic operations were implemented in Software, and hence performed on the primary CPU. The Signature Algorithm used in the group is EdDSA with curve Ed25519.

Memory utilization

Memory utilization was calculated by combining data from compiled ELF files with stack usage numbers from run-time. The GNU tool objdump was used to extract the size of the .Text, .BSS, and .Data segments from the ELF files. Also, a known bit pattern was written to the stack at system boot, to enable counting the number of bytes on the stack that the tested program had overwritten, after conclusion of one experiment run. The RAM utilization was then calculated as the size of .Data + .BSS + stack, while the ROM utilization was calculated as the size of .BSS + .Text.

Figure 4.2 shows the RAM and ROM utilization for the Zoul platform, additionally highlighting the stack usage and the memory overhead corresponding to the cryptographic operations.

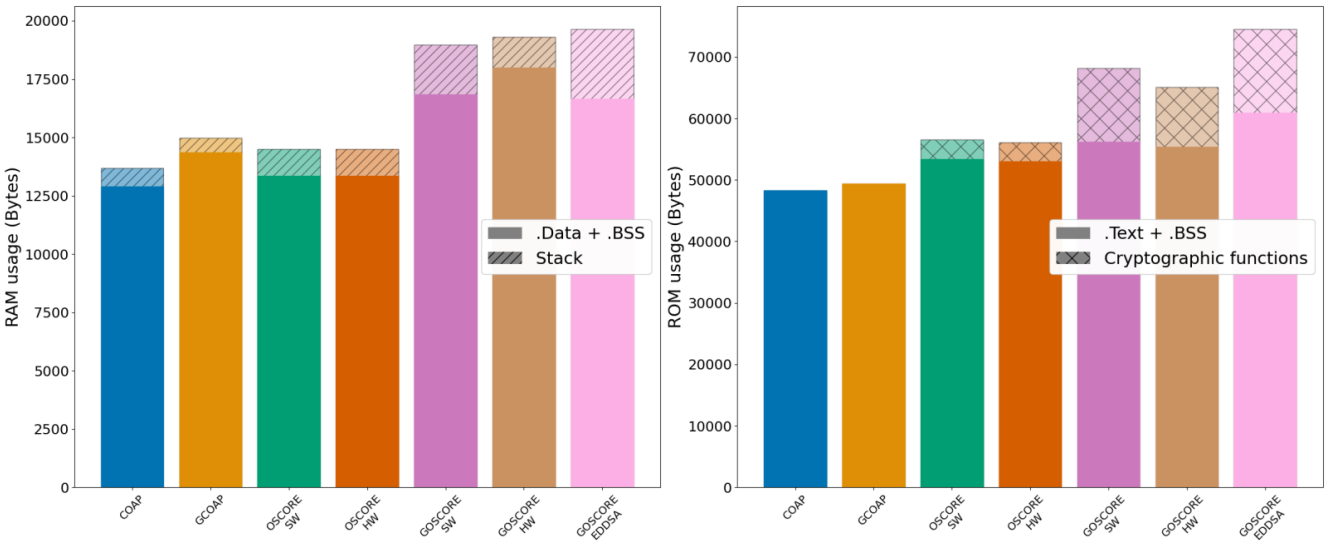


Figure 4.2 - Memory utilization for the Zoul platform, out of 32KB of RAM and 512KB of ROM

The slight increase in both RAM and ROM for Group-COAP compared to COAP is caused by the inclusion of IPv6 Multicast routing functionality. Additional routines for processing OSCORE messages add 2 KB of extra ROM and another 1 KB for the AES-CCM-128 and SHA256 algorithms. Although hardware-accelerated cryptography used in Group-OSCORE requires less ROM than the cryptographic software libraries, it requires more RAM. The code used to interface with the hardware accelerator on the Zoul platform implements ECDSA P-256 without any optimizations. This negatively impacts performance in terms of memory utilization. The software library used for ECDSA P-256, namely micro-ecc, is optimized to conserve memory. This explains how the hardware implementation of ECDSA P-256 surprisingly utilizes more memory. Finally, in the case of Group-OSCORE-EDDSA a larger RAM and ROM utilization was observed, which can be attributed to the used cryptographic library, which has been shown to require significant memory resources.

Figure 4.3 shows the memory utilization for the Simplelink platform. The comparison between software and hardware-accelerated cryptography solutions looks slightly different than in the case of the Zoul platform, which results from hardware discrepancies and the nuances of particular device driver implementations.

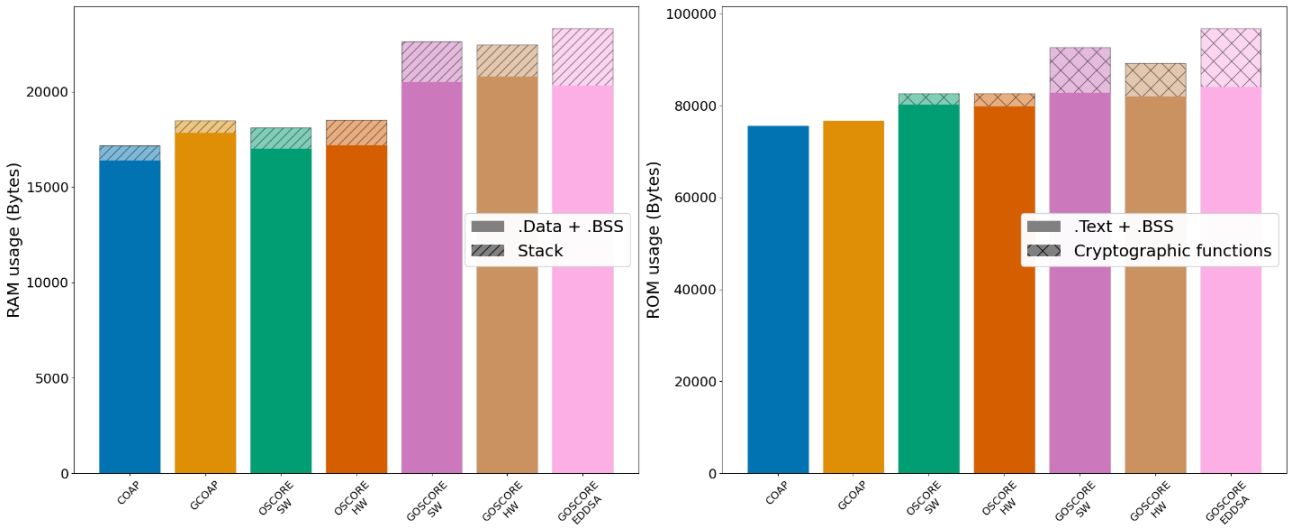


Figure 4.3 - Memory utilization for the Simplelink platform, out of 88KB of RAM and 608KB ROM

Since the Zoul platform has 32 KB of RAM and 512 KB of ROM, the increased amount of memory needed for Group OSCORE processing is reasonable. Thus, one can conclude that fitting Contiki-NG with a network stack and the Group OSCORE implementation in addition to application code is feasible on the Zoul platform. Also, low-power operation, resulting in only half of the Zoul RAM memory available, could be achieved in the presence of Group OSCORE. The Simplelink platform has even more memory, i.e., 88 KB of RAM and 608 KB of ROM. Thus, the increased memory utilization due to Group OSCORE is not significant in terms of total memory.

Round trip time

Round trip time (RTT) was also measured, as the time elapsed from when the processing of an outgoing request starts on the constrained client, until when a received response has been processed by the client. Varied sizes for the application payload were considered, excluding any message overhead from headers, AEAD-encryption tags, or signatures. Due to the use of response delay randomization required by CoAP group communication for avoiding collisions of responses, both the time elapsed until the first response arrived and the time elapsed until the last response arrived at the client were considered. As used together with group communication, Group-OSCORE also relies on the response delay randomization mechanism. When evaluating the test cases relying on group communication, the servers chose the delay of each response as a random value between 0 and 8 s, to conservatively prevent collisions and ensure a feasible collection of results. It follows that, on average, a large share of RTT values when group communication is used consists of the added randomized delay.

Figure 4.4 shows the RTT measured when considering the Zoul platform, presenting two sets of response times.

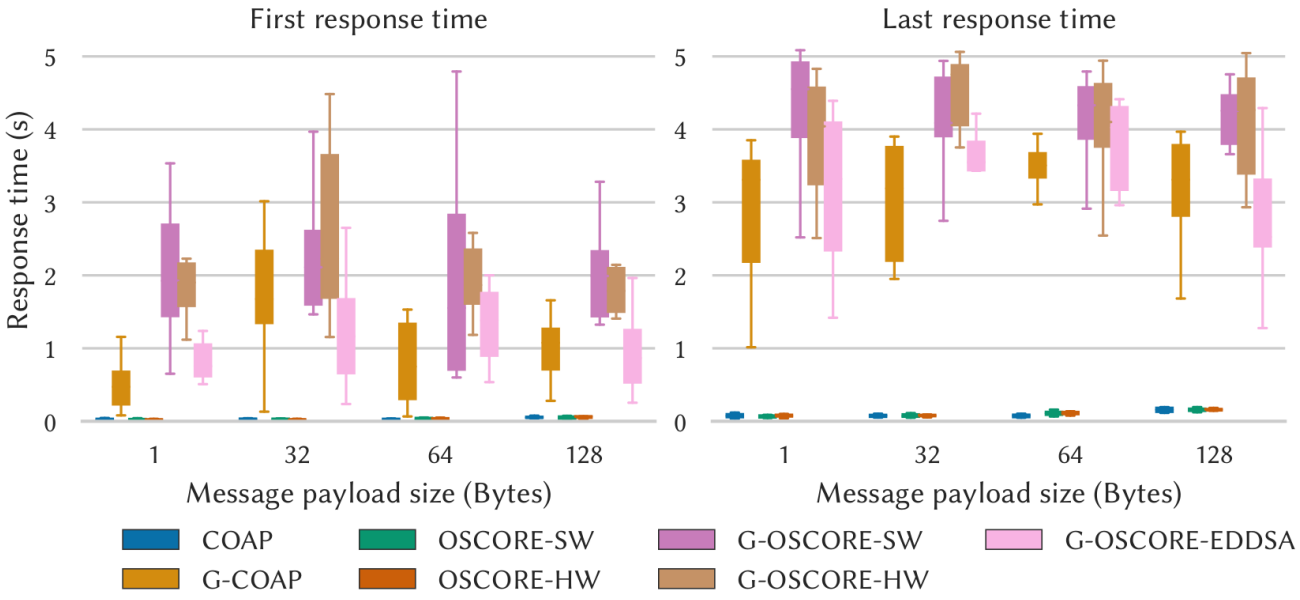


Figure 4.4 - Round Trip Time (RTT) measurements for the Zoul platform

The first set comprises response times for the three test cases COAP, OSCORE-SW and OSCORE-HW. Consistently with a quick response delivery, they are all low for both the first and the last response, with no notable influence from the message payload sizes considered. The second set comprises response times for the four test cases relying on group communication, i.e., Group-COAP, Group-OSCORE-SW, Group-OSCORE-HW and Group-OSCORE-EDDSA. In these cases, the response times are greater and fall within a broad range of values. This is mainly due to the random delay necessitated when using group communication for CoAP, as introduced by each responding server. The variability introduced by the random delays overshadows the possible role played by the message payload sizes. More in general, the range of values where the random delay is selected from can be determined according to the characteristics of the system and network deployment.

Figure 4.5 shows the RTT measured for the Simplelink platform, providing insights similar to those observed for the Zoul platform. As mentioned earlier in the analysis of the Zoul platform, the response times for the three test cases involving purely unicast communication are characterized by small, consistent, and comparable values. Similarly, in the case of the Simplelink platform, these test cases exhibit similar response time patterns, also being small and consistent.

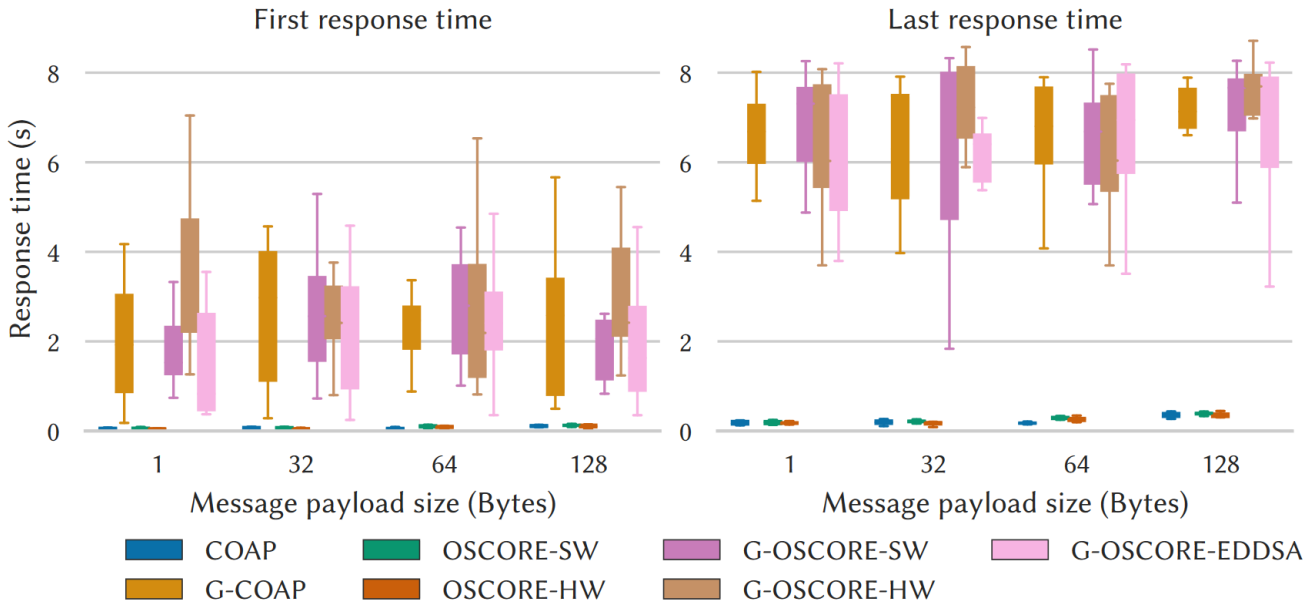


Figure 4.5 - Round Trip Time (RTT) measurements for the Simplelink platform

The four test cases relying on group communication for the Simplelink platform show a notable increase in response times compared to the unicast cases. This increase can be attributed to the random delays introduced by the servers involved in the group communication process. Just like for the Zoul platform, the variability in response times for group communication with the Simplelink platform is attributed to the random delays that are necessarily introduced when utilizing CoAP with group communication.

CPU usage time

In order to evaluate the impact of the cryptographic operations on the overall message processing, the time required to perform such functions was measured and compared against the total message processing time. For OSCORE, the time needed to encrypt (decrypt) a message was measured and presented as the percentage of the total time required to serialize an outgoing (parse an incoming) message. For Group-OSCORE, the time needed to encrypt and sign (decrypt and verify the signature of) an outgoing (incoming) message was measured and presented as the percentage of the total time needed to serialize an outgoing (parse an incoming) message.

Table 4.3 shows the processing times for the Zoul platform in milliseconds.

Payload (Bytes)	1		32		64		128	
	S	P	S	P	S	P	S	P
COAP	0.05	0.12	0.07	0.12	0.09	0.12	0.11	0.12
Group-COAP	0.05	0.12	0.05	0.12	0.06	0.12	0.06	0.12
OSCORE-SW	0.51	0.56	0.71	0.76	0.92	0.95	1.35	1.34
Encypt/Decrypt	54.2%	53.0%	65.0%	64.3%	72.1%	71.1%	77.1%	79.2%
OSCORE-HW	0.32	0.38	0.35	0.39	0.38	0.40	0.45	0.44
Encypt/Decrypt	26.7%	27.6%	29.3%	28.1%	28.8%	31.8%	30.4%	34.5%
G-OSCORE-SW	576.97	624.99	577.68	626.98	577.65	625.98	578.36	627.65
Encypt/Decrypt	99.9%	99.9%	99.8%	99.9%	99.9%	99.9%	99.9%	99.9%
G-OSCORE-HW	348.84	706.55	349.08	711.96	349.29	710.74	349.74	707.45
Encypt/Decrypt	99.9%	99.9%	99.8%	99.9%	99.8%	99.9%	99.8%	99.9%
G-OSCORE-EDDSA	107.50	269.03	109.13	268.09	109.81	269.15	111.88	271.39
Encypt/Decrypt	99.2%	99.8%	99.4%	99.8%	99.4%	99.8%	99.3%	99.8%

Table 4.3 - CPU usage of the Zoul platform in milliseconds, for serializing (S) and parsing (P) of messages with cryptographic operations as a percentage of total

The results in Table 4.3 show that hardware acceleration for cryptographic operations improves OSCORE performance, as hardware accelerated cryptography takes up a smaller fraction of the total processing time, i.e., less than 50%. When Group OSCORE is used, cryptographic functions are responsible for most of the time taken to serialize and parse messages. Particularly, over 99% of the processing time is spent for producing or verifying the signature of an outgoing or incoming message, respectively. Note that using hardware acceleration for cryptographic operations reduces the overall time needed to serialize messages. Instead, signature verification and the total message parsing time do not show the same performance improvement when hardware acceleration for cryptographic operations is applied.

Results for the Simplelink platform are shown in Table 4.4. The reported values follow the same trends as for the results for the Zoul platform above, but they are generally lower compared to the values for the Zoul platform. One notable exception concerns the results for the test case OSCORE-SW, where the times are longer for the Simplelink platform. This is due to the less efficient AES-CCM Simplelink implementation, rendered by the increased delay share of the OSCORE-SW crypto functions.

Payload (Bytes)	1		32		64		128	
	S	P	S	P	S	P	S	P
COAP	0.00	0.18	0.06	0.16	0.06	0.16	0.06	0.17
Group-COAP	0.03	0.20	0.05	0.20	0.06	0.20	0.07	0.21
OSCORE-SW	1.85	1.98	2.93	3.06	4.03	4.14	6.18	6.28
Encypt/Decrypt	90.8%	85.2%	93.8%	90.0%	94.5%	92.2%	96.2%	94.8%
OSCORE-HW	0.31	0.46	0.37	0.49	0.42	0.51	0.61	0.68
Encypt/Decrypt	41.2%	34.2%	50.0%	37.5%	43.5%	36.9%	51.0%	49.5%
G-OSCORE-SW	296.91	317.10	298.40	319.06	299.23	319.76	301.76	322.94
Encypt/Decrypt	99.8%	99.8%	99.8%	99.8%	99.8%	99.8%	99.8%	99.8%
G-OSCORE-HW	233.89	467.10	233.95	466.39	234.01	466.50	234.25	466.08
Encypt/Decrypt	99.8%	99.9%	99.8%	99.9%	99.7%	99.9%	99.7%	99.9%
G-OSCORE-EDDSA	50.58	112.17	52.63	112.92	53.92	114.14	57.23	117.58
Encypt/Decrypt	99.1%	99.5%	99.0%	99.4%	98.9%	99.4%	98.8%	99.4%

Table 4.4 - CPU usage of the SimpleLink platform in milliseconds, for serializing (S) and parsing (P) of messages with cryptographic operations as a percentage of total

The processing time results for Zoul and Simplelink platforms show similar trends, albeit shorter total times for the Simplelink platform can be observed. OSCORE cryptography takes between 25% and 90% when implemented in software and between 10% and 55% when implemented in hardware. This is a significant part of the processing time, but it is noticeably smaller compared to the 99% of the total time that cryptography takes up when Group OSCORE is used. As learned from the OSCORE experiments, 1-2 ms were needed to encrypt and decrypt a message, while the further increase in the processing delay is due to verifying a signature or generating one. Hardware acceleration shortens the total processing time for Group OSCORE. Ed25519 using the Monocypher library shows excellent performance in terms of speed on both platforms, outperforming ECDSA P-256 in both hardware and software on both platforms. This speed comes at the penalty of a larger memory footprint in ROM and the largest utilization of RAM of all the tested asymmetric cryptography implementations on both platforms. This confirms the expectation that asymmetric cryptography is considerably more time-consuming than symmetric cryptography on these types of constrained devices.

Energy consumption

For resource-constrained devices, energy consumption is an important metric to assess when evaluating protocol performance. To measure the energy consumption a DC energy analyzer was used, i.e., a tool for measuring voltage and current with a high precision at a very high sampling frequency. In particular, the Joulescope device was used to measure the power consumption of the tested software configurations on both hardware platforms. The Joulescope measures voltage and current supplied to the device, as evaluated at 2 million samples per second with nanoampere precision. The actual energy consumed during message processing was measured. As in the case of CPU-time measurements, the period between the start of application-layer processing of the incoming message until the message is delivered to the application was considered for the measurement interval. The total time spent by the application-layer to fully prepare and process an outgoing message was measured, including possible security operations.

Figure 4.6 shows the measured energy consumption for the Zoul platform. The Y-scale is logarithmic in order to better display the results.

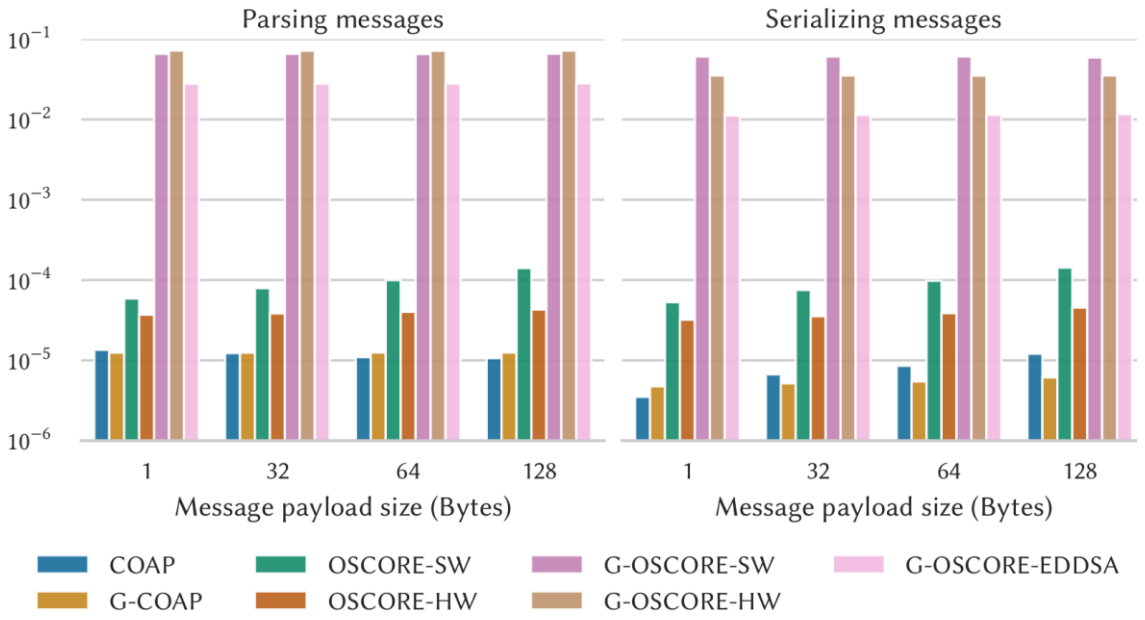


Figure 4.6 - Energy measurements for the Zoul platform

When analyzing the results for OSCORE-SW, OSCORE-HW, Group-OSCORE-SW and Group-OSCORE-HW, it is clear that the per-message energy consumption increases with growing message sizes for OSCORE-SW and OSCORE-HW, but remains virtually constant for COAP, Group-COAP and the three Group-OSCORE variants. This can be explained by the time needed to generate and verify signatures, which is almost constant. Furthermore, the energy consumption for Group-OSCORE-EDDSA is lower than that for the Group-OSCORE-SW and Group-OSCORE-HW results. This is due to the efficient (i.e., fast) cryptographic library used for EdDSA operations. As to COAP and Group-COAP, the roughly constant energy consumption can be explained by the noticeably short CPU time observed. For OSCORE-SW and OSCORE-HW, a linear growth trend can be observed, because of the increased processing time needed to encrypt the payload of the messages.

Figure 4.7 shows the results for the Simplelink platform. As for the previous platform, one can see that the test cases COAP, Group-COAP, Group-OSCORE-SW, Group-OSCORE-HW and Group-OSCORE-EDDSA show a roughly constant energy consumption across message payload sizes. Instead, the test cases OSCORE-SW and OSCORE-HW show a linear relation between message payload size and energy consumption. This is due to the encryption and decryption of the message payload taking more CPU-time, and thus energy, to process a larger payload. Among the three test cases relying on Group OSCORE, Group-OSCORE-SW displays the highest energy consumption for both parsing and serializing messages, while Group-OSCORE-HW and Group-OSCORE-EDDSA display the lowest energy consumption and a value in between, respectively.

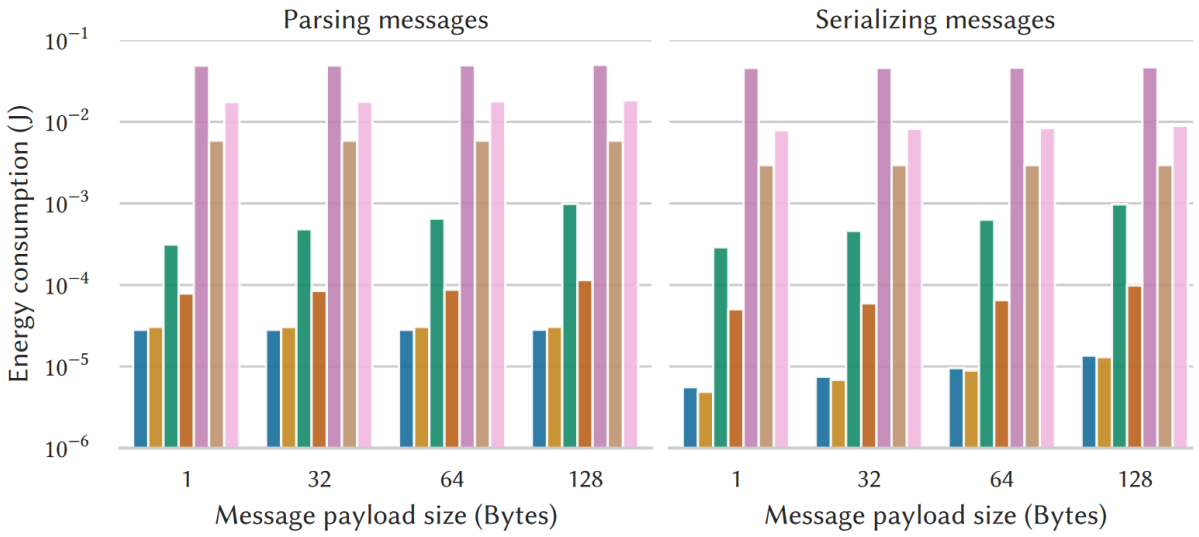


Figure 4.7 - Energy measurements for the Simplelink platform

For the Simplelink platform, the processing energy consumption per exchange is 2 mJ for OSCORE-SW, 96 mJ for Group-OSCORE-SW, 9 mJ for Group-OSCORE-HW, and 27 mJ for Group-OSCORE-EDDSA. For the Zoul platform, the per-exchange energy consumption is 0.3 mJ for OSCORE-SW, 125 mJ for Group-OSCORE-SW, 107 mJ for Group-OSCORE-HW, and 39 mJ for Group-OSCORE-EDDSA. Using OSCORE-SW as a baseline, Group-OSCORE-SW consumes 416 times more energy per message exchange. For Group-OSCORE-HW, the corresponding number is 356 times the energy of OSCORE-SW. Finally, Group-OSCORE-EDDSA consumes 130 times more energy per message exchange. This results in 48 times increase in energy consumption for Group-OSCORE-SW, 4.5 times increase for Group-OSCORE-HW, and 13.5 times increase for Group-OSCORE-EDDSA.

Summary

Overall, the performed experiments evaluated memory consumption, RTT, and energy consumption overhead of Group OSCORE, throughout a broad selection of alternative settings. The results show that incorporating Group OSCORE does not introduce significant RAM/ROM penalty on the tested platforms. However, Group OSCORE operations (most notably message signing and signature verification) contribute to a much larger RTT than CoAP, Group CoAP and OSCORE. Group OSCORE also consumes more energy per message sent if compared to OSCORE, which might be a limiting factor for constrained, battery powered IoT nodes communicating frequently. Nevertheless, the number of possible message exchanges is very high, also for battery-powered devices. Finally, one can notice that using the EdDSA signature algorithm resulted in better performance than ECDSA P-256, especially with respect to the ECDSA P-256 Software implementation.

As key takeaways from the performance evaluation, the memory utilization in terms of RAM and ROM is manageable for constrained IoT devices. The use of ECC signatures adds a non-negligible delay to a full message exchange and significantly contributes to energy consumption. However, the EdDSA signature algorithm displays considerably better performance than ECDSA P-256. Even though the complexity of ECC cryptography adds such penalties, Group OSCORE remains feasible for low-power applications.

5 Part 2 – Access and Usage Control for Server Resources

This section presents the developed security solutions within the area "Access and Usage Control for Server Resources".

5.1 *OSCORE Profile of ACE*

The solutions and methods presented in this section pertain to the following requirements defined in [D1.2]:

- *Non-Functional Requirements: PE-28, PE-29*
- *Security Requirements: SE-19, SE-30, SE-35, SE-40*

The solutions and methods presented in this section pertain to the following components defined in [D1.4]:

- *The "Authentication Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Key Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Secure Message Exchange Manager" component of the "Secure Communication Layer" module.*
- *The "Content Distribution Manager" component of the "Secure Communication Layer" module.*

This section overviews a security profile of the ACE framework for authentication and authorization (see Sections 3.8 and 3.8.3), i.e., the OSCORE profile, which has been published as the IETF Proposed Standard RFC 9203 [PAL21].

The OSCORE profile of ACE describes how a Client (C), and a Resource Server (RS) can engage in the ACE workflow and establish an OSCORE Security Context for securely communicating with one another using the OSCORE security protocol [SEL19] (see Section 3.7). Also, it allows C to re-establish an OSCORE Security Context with the RS using the same Access Token, as well as to update its own access rights at the RS, by means of a new, smaller Access Token.

Upon receiving a first Access Token request from C, the Authorization Server (AS) generates an OSCORE Security Context Object. This includes information and parameters for C and the RS to establish an OSCORE Security Context, such as and especially an OSCORE Master Secret and an Input Material Identifier. The AS includes the OSCORE Security Context Object into the Access Token to be released. After that, the AS provides C with both the Access Token and the OSCORE Security Context Object. For the sake of proof-of-possession, C has to prove to the RS to also possess the OSCORE Master Secret specified in the Access Token.
This first Access Token that C asks for accessing resources at the RS is denoted as the first Access Token of the first *token series* related to C and the RS.

Upon uploading the Access Token to the RS, both C and the RS exchange a pair of nonces as well as the respective OSCORE identifiers that they intend to use. Specifically, C provides the RS with the Access Token, a nonce N1, and an ID1 that it intends to use as its own OSCORE Recipient ID with the RS. If the RS successfully verifies the Access Token as valid and issued by the AS, then the RS replies to C by providing a nonce N2 and an ID2 that it intends to use as its own OSCORE Recipient ID with C. This exchange between C and the RS is not protected.

Then, C and RS use such values together with the OSCORE Security Context Object received from the AS, and derive a complete, fresh OSCORE Security Context. In particular, ID1 and ID2 are used as Recipient ID of C and the RS, respectively; the Master Secret and ID Context are as indicated in the OSCORE Security Context Object from the AS; the Master Salt is the concatenation of N1, N2, and an input salt value indicated in the OSCORE Security Context Object from the AS.

After that, C can send a first secure request to the RS, protected with the new OSCORE Security Context. Proof-of-possession is achieved when the RS receives such a first request and verifies it as cryptographically correct. Then, C and the RS can continue communicating by protecting the exchanged messages with their OSCORE Security Context, where the access of C to resources at the RS will be consistent with the authorization information specified in the Access Token stored at the RS.

If the Access Token becomes invalid (e.g., it expires), C can request a new one to the AS by sending a new Access Token request to the AS. The new Access Token will be the first Access Token in a new token series.

Instead, as long as the Access Token is valid, C can also proceed according to the two following options.

Option 1: Update of OSCORE Security Context. C can re-upload the same Access Token to the RS, in order to establish a new OSCORE Security Context with the RS.
This can be the case if the RS had to delete the stored Access Token (e.g., due to memory limitations), and thus denies access to C and replies with 4.01 (Unauthorized) error responses to attempted accesses from C.
Alternatively, C may want to switch to new keying material to use with the RS, by establishing a new OSCORE Security Context that supersedes the current one. In either case, C and the RS perform the same unprotected exchange discussed above, and provide each other with new N1, N2, ID1, and ID2 values used for deriving the new OSCORE Security Context according with the same OSCORE Security Context Object from the AS.
After that, C and the RS rely on the new OSCORE Security Context to protect their communications, and accesses of C to the resources at the RS still have to be consistent with the authorization information specified in the Access Token stored at the RS.

Option 2: Update of access rights. C may want to update its access rights for accessing resources at the RS, while preserving the same OSCORE Security Context with the RS.
To this end, C requests to the AS a new Access Token within the same token series. In order to do that, the Access Token request to the AS includes the new set of requested permissions to access the resources at the RS, and the value of the Input Material Identifier that the AS specified in the first Access Token of the token series (see above). The latter allows the AS to identify the token series in question, hence to issue a new Access Token which belongs to the same token series and is smaller in size. In particular, neither the new Access Token nor the response to C conveying it include the same OSCORE Security Context Object in full. Instead of that, the AS specifies again the same Input Material Identifier associated with the token series in question.
Once received the new Access Token, C uploads it to the RS. However, instead of relying on the unprotected exchange discussed above, C uploads the new Access Token to the RS by means of a message protected with the current OSCORE Security Context shared with the RS, and without specifying any additional information such as a nonce or an OSCORE Recipient ID. If the RS successfully verifies the message and the new Access Token, the RS stores the new Access Token, which supersedes the current one. At the same time, a new OSCORE Security Context is not established between C and the RS.
After that, C and the RS use the same, current OSCORE Security Context to protect their communications, while accesses of C to the resources at the RS will have to be consistent with the authorization information specified in the newly uploaded Access Token belonging to the same token series as the superseded one.

Figure 5.1 shows the different alternatives available per the workflow of the OSCORE profile discussed above, by taking the point of view of C as interacting with a fixed RS.

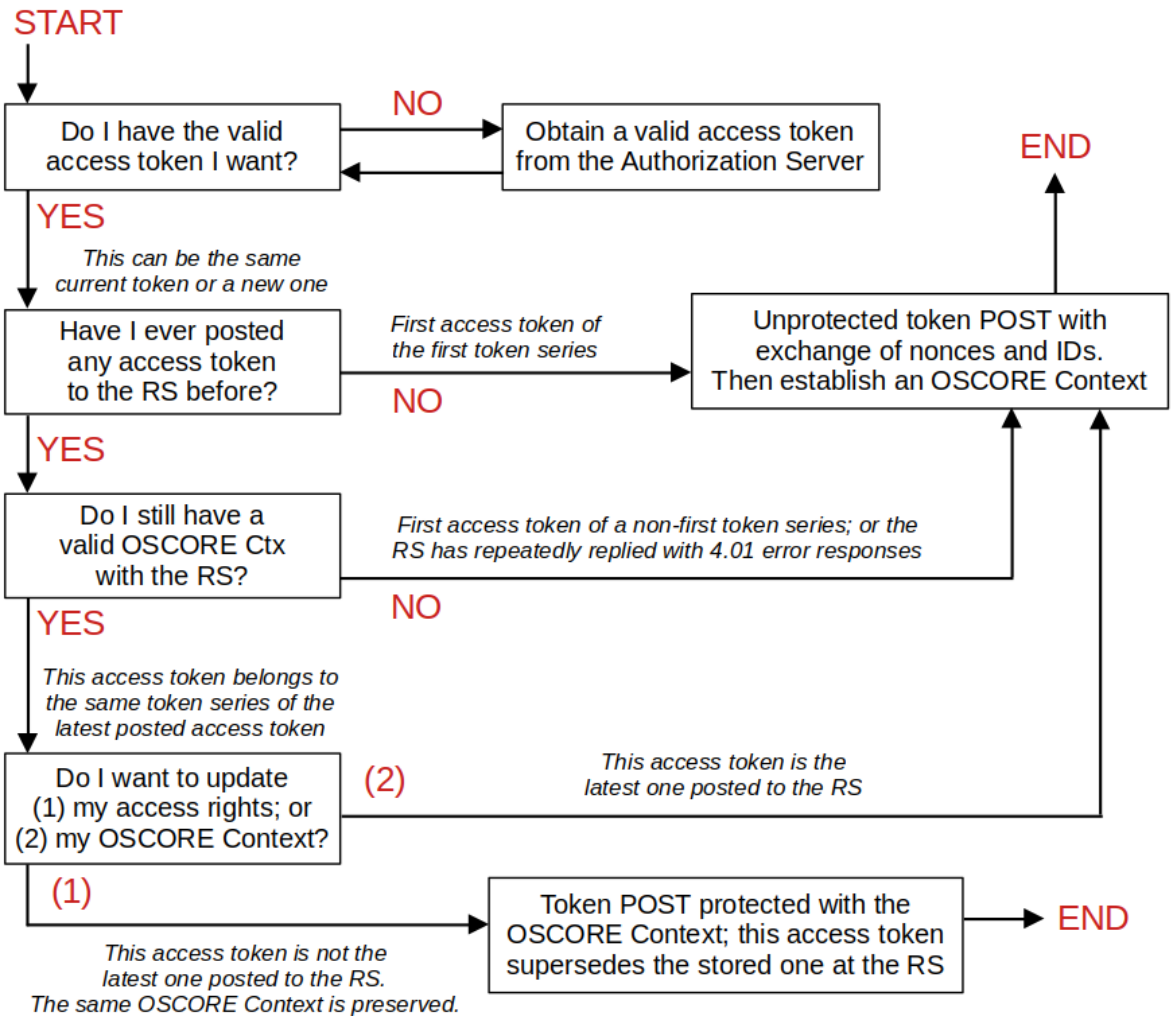


Figure 5.1 – Overview of the workflow of the OSCORE profile of ACE (point of view of C).

The OSCORE profile of ACE is also included in the Java implementation of the ACE framework from RISE available at [ACE-DEV], as available for use in the SIFIS-Home project. In the project, this has been especially used to establish OSCORE secure associations between CoAP endpoints that want to join an OSCORE group and the responsible Group Manager (see Section 6.1). This was first included in the focused WP3 demo overviewed in Annex B, and then integrated in the SIFIS-Home solution in the context of WP5 and WP6.

Notably, additional design and development activities have also been ongoing within the SIFIS-Home project to define two further security profiles of ACE. Due to time constraints and prioritization choices, these results have still not reached a maturity level that could enable their integration in the SIFIS-Home solution. For information, those activities are shortly summarized below.

First, the Group OSCORE profile [TIL23g] makes it possible to enforce fine-grained access control for operations that Clients within a group performs at Resource Servers that are members of that same group, while using the Group OSCORE security protocol to protect message exchanges (see Section 4.1). Second, the EDHOC and OSCORE profile of ACE [SEL23b] builds on the OSCORE profile presented in this section, with the difference that C and the RS establish their OSCORE Security Context by executing the EDHOC key establishment protocol (see Section 6.2), thus benefiting of its security properties such as forward secrecy of the established keying material and peer mutual authentication by means of public authentication credentials.

Notification of Revoked Access Credentials

The solutions and methods presented in this section pertain to the following requirements defined in [D1.2]:

- *Non-Functional Requirements: PE-28, PE-29*
- *Security Requirements: SE-19, SE-40*

5.2 The solutions and methods presented in this section pertain to the following components defined in [D1.4]:

- *The "Authentication Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Secure Message Exchange Manager" component of the "Secure Communication Layer" module.*
- *The "Content Distribution Manager" component of the "Secure Communication Layer" module.*

Access Tokens issued by an Authorization Server (AS) are meant to eventually expire. The AS can give an explicit indication of expiration time to a Resource Server (as a piece of information within the Access Token) as well as to a Client (C) when providing the Access Token to C. Then, when the Access Token expires, both C and the RS discard it, and terminate their secure association previously built based on such an Access Token. After that, C can request a new Access Token from the AS, and then upload it to the RS, in order to establish a new secure association with the RS and resume accessing protected resources at the RS, as per the new Access Token.

On top of that, there are additional circumstances when the AS may revoke Access Token, before its expiration time comes. Practical effects should be the same ones mentioned above for the case of expiration, and they apply to both C and the RS, hereafter referred to as registered devices, due to their registration at the AS.

Examples of situations resulting in revoking an Access Token include:

- a registered device has been decommissioned;
- a registered device has been compromised, or it is suspected of being compromised;
- there has been a change in the ACE profile for a registered device;
- there has been a change in the access policies for a registered device;
- there has been a change in the outcome of policy evaluation for a registered device (e.g., if policy assessment depends on dynamic conditions in the execution environment, the user context, or the resource utilization).

In the OAuth 2.0 framework [HAR12], it is possible for C to initiate the revocation of an Access Token, as specified in [LOD13]. This builds on the assumption that, in OAuth 2.0, the AS issues Access Tokens with a relatively short lifetime. However, this is likely not the case for the AS in the ACE framework. In fact, resource-constrained and intermittently connected devices practically require Access Tokens with relatively long lifetimes.

With particular reference to the ACE framework (see Section 3.8), an RS would be able to learn about revoked Access Tokens that it owns, by checking at the AS through the introspection mechanism (see Section 3.8.2), in case the AS provides such an optional service. In such a case, requests from introspection at the AS are limited to checking one Access Token at the time. On the other hand, C has no means to learn whether any of the Access Tokens it owns has been revoked.

More generally, it is not possible for the AS to take the initiative and notify registered devices about pertaining Access Tokens that have been revoked, but are not expired yet. Specifically, an Access Token pertains to a Client if the AS has issued the Access Token and provided it to that Client. Also, an Access Token pertains to a Resource

Server if the AS has issued the Access Token to be consumed by that Resource Server.

The novel approach presented in this section and specified in the standardization proposal [TIL23h] aims to fill this gap. That is, it specifies a method for registered devices to access and optionally observe a Token Revocation List (TRL) resource at the AS, in order to obtain an updated list of revoked, but yet not expired, pertaining Access Tokens.

In particular, registered devices can rely on resource observation [HAR15] for CoAP [SHE14]. That is, the AS would automatically send a notification to an observer registered device, when the status of the TRL resource changes. Specifically, this happens when:

- an Access Token pertaining to that device gets revoked; or
- a revoked Access Token previously included in the list eventually expires.

The main benefits of this method are that it assists both Clients and Resource Servers, it complements the introspection mechanism, and it does not require any additional resources or endpoints to be created on the registered devices.

The TRL resource at the AS does not contain the full representation of the currently revoked Access Tokens. Instead, it contains their ad-hoc identifiers computed for this purpose. Such identifiers, namely token hashes, are computed as per [FAR13], and make it possible to correctly handle different types of Access Tokens conveyed over different transports.

As mentioned above, a registered device can at any time send a request to the TRL resource at the AS, or observe it to get automatic notification responses in case of changes in the TRL. In either case, the registered device is not going to receive the full content of the TRL, but rather only a pertaining subset, which contains only token hashes of Access Tokens pertaining to that registered device, as extracted from the whole current content of the TRL resource.

More specifically, a registered device can access the TRL resource at the AS in two different modes, which result in different responses from the AS. As described above, the interactions with the AS can be based on a single request-response exchange, or rather based on CoAP observations, where the AS sends additional notification responses to an observer registered device, upon changes in the TRL resource that pertain to that registered device. The following provides an overview of such two modes of operation.

Full query mode – This mode of operation must be supported by the AS. When using this mode, a registered device relies on a GET request to the TRL resource at the AS, with no additional query parameters specified.

The response from the AS includes a parameter "full_set", which specifies the token hashes of the revoked Access Tokens currently in the TRL and pertaining to the requester registered device.

Diff query mode – This mode of operation may be supported by the AS. When using this mode, a registered device relies on a GET request to the TRL resource at the AS. The request additionally includes the query parameter "diff", specifying an unsigned integer value N. Then, the AS provides the requester register device with a set of TRL diff entries, which represent updates occurred to the TRL subset pertaining to that register device.

In particular, each returned diff entry is related to one of the N most recent updates in the subset of the TRL pertaining to the requester registered device. The entry associated with one of such updates contains a list of token hashes, such that the following holds: i) the corresponding revoked Access Tokens pertain to the requester

registered device; and ii) at the update associated with the diff entry in question, they were added to the TRL (as revoked) or removed from the TRL (as eventually expired following their earlier revocation).

The diff query mode can additionally rely on its "Cursor" extension, which the AS can also optionally support. If the diff query mode with the "Cursor" extension is used, a response from the AS also provides the requester registered device with: i) a parameter "cursor", specifying an unsigned integer value that identifies the most recent update to the TRL conveyed as a diff entry in the present response; and ii) a boolean parameter "more", specifying whether further updates have occurred to the TRL resource as pertaining to the requester registered device, after the most recent update specified in the present response.

If the registered device also supports the "Cursor" extension, this effectively allows the registered device to retrieve a set of diff entries not only as the most recent TRL updates, but also starting from after the cursor value specified by the AS and used as resumption point. To this end, a registered device can send to the AS a request in diff query mode, and additionally specify the value of the received "cursor" parameter as value of a "cursor" query parameter. In turn, this effectively allows the AS to provide large sets of diff entries in smaller chunks, as well as the registered device to quickly get itself aligned with updates to the TRL resource after a long time during which no updated information has been received from the AS.

Irrespective of the used mode of operation, a registered device expunges every stored Access Token associated with a token hash specified in a response from the AS. Furthermore, the following also applies if the registered device is specifically a RS: the RS must store the token hashes received from the AS; the RS must not accept or store an access token, in case the corresponding token hash is among the currently stored ones. It is safe for the RS to delete a stored token hash th1 associated with an Access Token t1, when both the following two conditions hold: the RS has received and seen t1, irrespective of having accepted and stored it; and the RS has gained knowledge that t1 has expired, which can have relied on different approaches.

An implementation of the solution presented in this section from CNR is available at [ACE-UCON-DEV], as building on the implementation of the ACE framework from RISE available at [ACE-DEV]. Furthermore, the solution presented in this section has been used as a building block towards the combined enforcement of access and usage control (see Section 5.4), which in turn has been integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 and the Smart-Home pilot within WP6.

5.3 *Usage Control Framework*

The solutions and methods presented in this section pertain to the following requirements defined in [D1.2]:

- *Functional Requirements: F-30, F-31, F-32, F-33, F-34, F-35, F-47, F-48, F-49, F-50, F-51, F-52, F-53*
- *Non-Functional Requirements: P-19, PE-20, PE-21, PE-28, PE-29, US-15, US-16*
- *Security Requirements: SE-19, SE-40, SE-42*

The solutions and methods presented in this section pertain to the following components defined in [D1.4]:

- *The "Authentication Manager" component of the "Secure Lifecycle Manager" module.*

The *Usage Control (UCON) Framework* regulates the usage of a resource following the UCON model [PAR04] (see Section 3.9). The UCON framework architecture extends the XACML reference architecture [XAC13] to include the components required for the evaluation of XACML-based *Usage Control Policies* (UCPs) and for

the management and revocation of ongoing usage sessions because of mutable attributes.

A UCP is a policy written in the UPOL policy language [DIC18] and is composed of three different sections, i.e., pre-, on- and post- sections, which are evaluated separately and at different times. A standalone XACML policy is derived from each section, and derived policies can be evaluated by a traditional XACML engine. The first policy is called *pre-policy* and contains pre-authorizations, pre-conditions, and pre-obligations; the second policy is called *on-policy* and contains on-authorizations, on-conditions, and on-obligations; and the third policy is called *post-policy* and contains post-authorizations, post-conditions, and post-obligations.

Ongoing usage sessions are continuously monitored in the context of the on-policy. As soon as the value of a mutable attribute changes, a *policy re-evaluation* starts to verify whether the access is still legit or should be revoked, i.e., whether the on-policy is still satisfied or not after the attribute value has changed.

The **Usage Control System (UCS)** is the core of the UCON framework. The latter includes other components that interact with the UCS. Figure 5.2 shows the UCON framework architecture and its main components, which are described in the following.

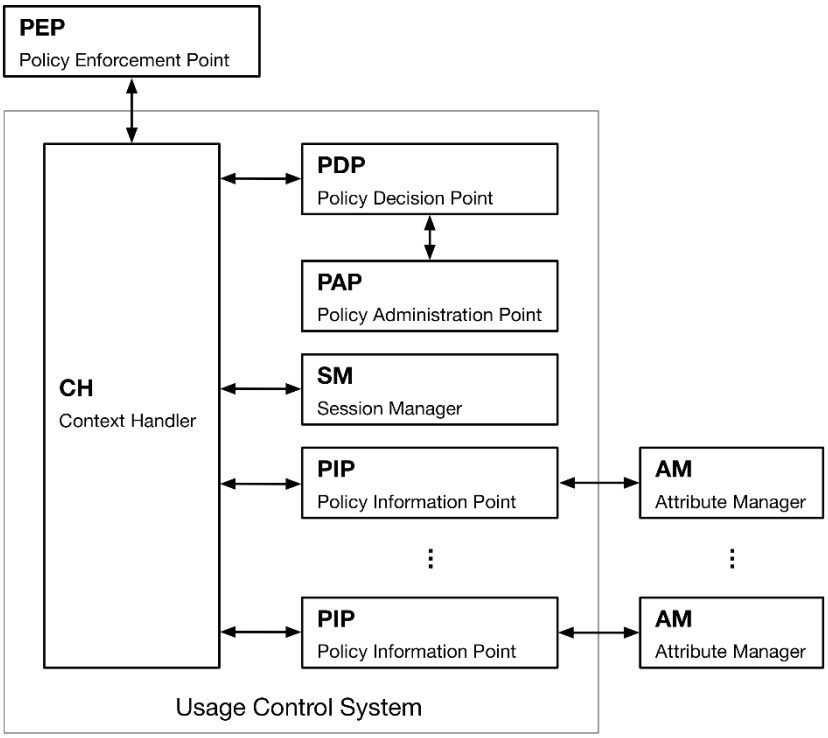


Figure 5.2 - UCON framework architecture

The **Policy Enforcement Point (PEP)** component regulates the access to a resource for a subject following the instructions received by the UCS. The PEP acts on behalf of the user that intends to access a resource and interacts with the UCS through three different types of messages: *tryAccess*, *startAccess*, and *endAccess*.

- **tryAccess** message: the PEP gathers (i) information about the *subject* performing the access request, e.g., their identity, role, etc., (ii) the *action* to be performed, e.g., read, write, etc., (iii) the *resource* to be accessed, e.g., a file, a thing, etc., and (iv) available *environmental information*, e.g., some sensor's reading, weather conditions, current day and time, etc.

  By using this information, the PEP creates and sends a tryAccess message containing an XACML request

(*access request*) to the UCS for evaluation (step 1 in Figure 5.3). More in detail, the request reaches the Context Handler (CH) –the front-end of the UCS– which manipulates it and asks the Policy Decision Point (PDP) to evaluate it against the pre-policy.

After evaluation, the CH replies with either a positive response (*permitAccess* response) or a negative response (*denyAccess* response), shown in step 2 of Figure 5.3. If a positive response is received, the PEP authorizes the subject *s* to access the resource *r* to perform the action *a* on it.

- **startAccess** message: upon receiving a permitAccess response following a tryAccess message, the PEP sends a startAccess message to the CH as soon as the access has started. In this phase, the PEP does not specify a new access request, and the PDP evaluates the original XACML request against the on-policy.

  After evaluation, the CH replies with either a positive response (permitAccess response) or a negative response (*revokeAccess* response), shown in step 4 of Figure 5.3. If a negative response is received, the PEP terminates the access to the resource *r* for the subject *s* and sends an endAccess message to the CH.

- **endAccess** message: the PEP sends this message to the UCS as soon as the access to the resource is terminated. An access can be terminated for two different reasons: (i) the access has naturally ended (step 5(i) of Figure 5.3), or (ii) the PEP received a revokeAccess message from the UCS (step 5(ii) of Figure 5.3). In the latter case, upon receiving the revokeAccess message, the PEP first undertakes actions to terminate the access and then informs the UCS through an endAccess message (step 6(ii) of Figure 5.3).

  In both scenarios, the PDP evaluates the original XACML request against the post-policy, which typically includes obligations (see Section 3.9). After evaluation, the CH replies with either a positive response (permitAccess response) or a negative response (denyAccess response), shown in steps 6(i) and 7(ii) of Figure 5.3. Note that, independently of the UCS response, the access to the resource terminated before the endAccess message was sent by the PEP.

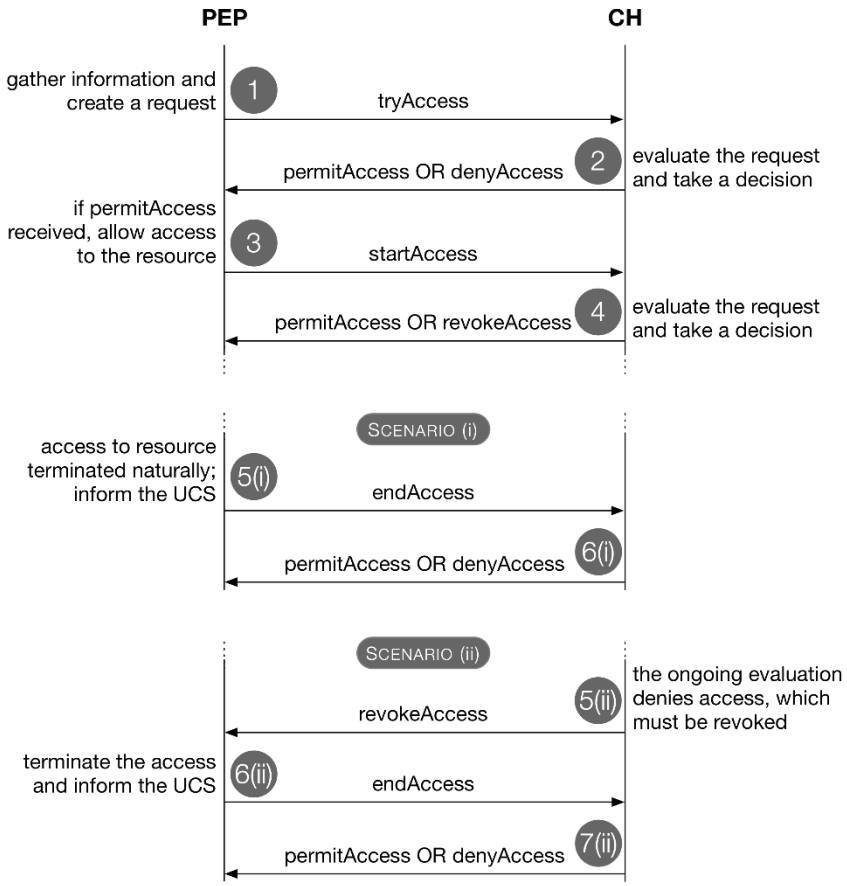


Figure 5.3 - PEP-CH interaction

The **Policy Administration Point (PAP)** component manages and stores the UCPs. Each UCP must be written in the UPOL language and must implement the pre- and the on- sections, and, optionally, the post- section.

The **Policy Decision Point (PDP)** component evaluates an access request against an access policy. When serving a tryAccess message, the PDP is also responsible for finding an *applicable UCP* UCP* among those stored at the PAP to be used for evaluation. An applicable UCP is a policy whose XACML <Target> field matches the access request, and, therefore, the access request can be evaluated against.

The PDP evaluates either the pre-policy, the on-policy, or the post-policy of the applicable UCP against the access request and produces an *access decision*. The access decision is the result of the evaluation, and it is either Permit –if the policy is satisfied by the request– or Deny otherwise. For the sake of simplicity, the results Indeterminate and NotApplicable are omitted here.

Both the access request and the access policy are expressed in XACML format, thus the PDP can use a standard XACML engine, such as WSO2 Balana [BAL21], for evaluation.

The **Attribute Managers (AMs)** are components responsible for mutable attributes. They communicate with the Policy Information Points (PIPs) and provide them with fresh attribute values. AMs can also manage immutable attributes and always provide the PIPs with the same value.

Examples of AMs can be local and remote databases, a file stored on the file system, a resource reachable at a

URL, or an Identity Provider controlling users' information, such as nationality or age. These can be mutable attributes since their value can change over time.

The **Policy Information Points (PIPs)** are adapters placed between CH and AMs, and their duty is to provide the CH with fresh attribute values. They offer a standard interface to the CH, while the interface with AMs is PIP specific.

The PIP-CH interface consists of four methods: (i) *retrieve*, (ii) *subscribe*, (iii) *unsubscribe*, and (iv) *update*. The retrieve method is invoked by the CH to obtain fresh attribute values for the attributes the PIP is responsible for. When calling this method, the CH can specify a value, e.g., an identity number, that the PIP uses to query the AM to obtain the pertaining attribute values. With reference to the previous example, the PIP could send the identity number to the AM, which returns the age associated with that number.

The subscribe method is invoked by the CH to get notified when an attribute value changes. When a PIP receives a subscription request, it starts a continuous monitoring of the attribute at the AM. As soon as the attribute value changes, the PIP notifies the CH, which performs a policy re-evaluation.

The unsubscribe method is invoked by the CH to stop receiving notifications from a PIP. When a PIP receives a request for subscription cancellation, it stops the attribute monitoring.

The update method is invoked by the CH to change the value of an attribute at the AM. When a PIP receives an update request, it sends a request to the AM in order to update the attribute value with the one provided by the CH.

The PIP-AM interface is specific for every PIP since AMs are heterogeneous components. The way in which the PIP retrieves an attribute value is clearly dependent on the specific AM it is attached to. Examples are resource polling at the AM, subscription to a publish-subscribe topic resource at the AM, and resource observation through the CoAP Observe Option [RFC7252][RFC7641].

The **Context Handler (CH)** component interacts with the PEP according to the protocol shown in Figure 5.3 and coordinates the process of evaluation of access requests.

For any type of message received from the PEP, as well as for performing a policy re-evaluation, the CH retrieves fresh attribute values from the PIPs, and it forms an *enriched request* by adding these attributes and their values to the original XACML access request.

Then, the CH asks the PDP to find an applicable UCP, namely UCP*. Finally, the CH sends the enriched request and either the pre-policy, the on-policy or the post-policy of UCP* to the PDP, which produces an access decision. The access decision is sent back to the CH, which undertakes different actions depending on the type of message it is serving:

tryAccess message:

- Deny – The CH sends a denyAccess response to the PEP.
- Permit – The CH communicates to the Session Manager (SM) that a new session for the current request must be created. The session includes a unique identifier (*SessionID*), the original XACML access request, UCP*, and the *status* of the access, which in this case is TRY_ACCESS. Then, the CH includes the SessionID in a permitAccess response and sends it to the PEP.

startAccess message:

- Deny – The CH communicates the SessionID to the SM, which updates the related session with the status REVOKE_ACCESS. Then, the CH sends a revokeAccess response to the PEP.

- Permit – The CH communicates the SessionID to the SM, which updates the related session with the status START_ACCESS. Then, the CH sends a permitAccess response to the PEP. From that moment on, the mutable attributes in the on-policy are continuously monitored: the CH subscribes to the pertaining PIPs, which notify it in the event of attribute value change. When this happens, the CH performs a policy re-evaluation.

policy re-evaluation:

- Deny – The CH communicates the SessionID to the SM, which updates the related session with the status REVOKE_ACCESS. Then, the CH cancels its subscription to the pertaining PIPs and sends a revokeAccess response to the PEP.
- Permit – The CH performs no further action.

endAccess message:

- Deny – The CH communicates the SessionID to the SM, which deletes the related session. Then, the CH sends a denyAccess response to the PEP.
- Permit – The CH communicates the SessionID to the SM, which deletes the related session. Then, the CH sends a permitAccess response to the PEP.

The **Session Manager (SM)** component keeps track and administers the lifecycle of usage control sessions. A session is the representation of an existing access. It is created when the access is first granted and is deleted after the access has terminated. A session consists of at least the following information:

- The session identifier (SessionID), i.e., a unique label that identifies an exact session;
- The status, which identifies the current state of an access and can assume the value TRY_ACCESS, START_ACCESS, or REVOKE_ACCESS;
- The original XACML access request; and
- The UCP against which the access request was evaluated.

The SM creates a new session when the access decision following a tryAccess message is Permit; it updates the session after the evaluation of a startAccess message or if the access decision following a policy re-evaluation is Deny; and it deletes the session after the evaluation of an endAccess message.

The SM is queried by the CH every time it gets notified by a PIP of an attribute value change. When this happens, the SM retrieves and sends back to the CH the SessionIDs of the *affected sessions*. These are the sessions whose status is equal to START_ACCESS and whose on-policy includes the attribute that has changed. Then, the CH performs a policy re-evaluation for all the affected sessions. The sessions for which the access decision after re-evaluation is Deny are updated with status REVOKE_ACCESS.

The solution presented in this section has been used as a building block towards the combined enforcement of access and usage control (see Section 5.4), which in turn has been integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 and the Smart-Home pilot within WP6.

5.4

Combined Enforcement of Access and Usage Control

The solutions and methods presented in this section pertain to the following requirements defined in [D1.2]:

- *Non-Functional Requirements: PE-28, PE-29*
- *Security Requirements: SE-19, SE-40*

The solutions and methods presented in this section pertain to the following components defined in [D1.4]:

- *The "Authentication Manager" component of the "Secure Lifecycle Manager" module.*

- *The "Secure Message Exchange Manager" component of the "Secure Communication Layer" module.*
- *The "Content Distribution Manager" component of the "Secure Communication Layer" module.*

When using the ACE framework to enforce access control (see Section 3.8), the Authorization Server (AS) must implement an evaluation and decision process to determine if an Access Token can be issued to a Client asking for access to resources at a target Resource Server (RS), and with what exact scope. In addition, it would be good for the AS to be able to perform a dynamic assessment of access control policies, possibly resulting in the revocation of issued Access Tokens before their expiration. To this end, the AS can leverage the UCON model (see Section 3.9), by relying on an integrated and specifically customized UCON-based decision maker (see Section 5.3). The following describes how such a decision maker component has been integrated into the AS.

A Java implementation from CNR [ACE-UCON-DEV] integrates a customized UCON-based decision maker within the implementation of the ACE framework from RISE [ACE-DEV], together with the mechanism for automatic notification of revoked Access Tokens described in Section 5.2. This has also been integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 and the Smart-Home pilot within WP6.

5.4.1 Integration of the UCON Framework into the ACE Framework

The PEP component is embedded in the AS and interacts with the /token endpoint, which entrusts it the practical task of determining the rights to be granted in accessing the set of resources at the RS specified by the Client. In order to do this, the PEP communicates with the UCS through the mechanisms described in Section 5.3.

In ACE, a Client C performing an Access Token request to the /token endpoint at the AS specifies an audience AUD and a scope SCOPE. If not specified, a default audience and scope are assumed. Through this request, C is essentially saying that it wants to access a specific resource RES (or more than one, as inferred from the scope) at a target RS and to perform an operation OP on that resource. Note that all the ACE actors are aware of the semantics used to express a scope and are thus able to map scopes to resources and operations on those. For example, a scope named "r_temp" might be mapped to the resource "temp_sensor" and operation "read".

From a UCON perspective, the former example translates to: the subject C wants to access the resource RES and perform the action OP on it. The target RS can be specified as an additional attribute both in the XACML requests and in the UCPs. An Access Token request can therefore be transformed by the PEP into an XACML request that the UCS can evaluate. An example of XACML request is reported in Figure 5.4.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Request ReturnPolicyIdList="false" CombinedDecision="false"
    xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">C</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">RES</AttributeValue>
        </Attribute>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-server" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">AUD</AttributeValue>
        </Attribute>
    </Attributes>
        <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">OP</AttributeValue>
        </Attribute>
    </Attributes>
</Request>
```

Figure 5.4 - Example of XACML access request. The Client C has requested access privileges to perform the

operation OP on the resource RES at the audience AUD associated with the target RS.

In the ACE framework, the scope requested by C is actually mapped into a set of resources and operations on those. For example, the scope SCOPE could be mapped to the set of resources and operations {<RES1, OP1>, <RES2, OP1>}. If it is possible to satisfy the request only partially, the AS grants to C only a subset of the requested access rights on such resources and operations. Consistently with a correct integration of the UCON framework into ACE, the AS first determines the set of resources and requested operations on those from the scope specified by C. Then, the PEP creates a separate XACML access request for each resource as described above. Such requests are individually submitted in the form of tryAccess messages to the UCS, which evaluates them and creates the sessions at the SM for those whose access decision is Permit. After that, the UCS returns either a permitAccess or a denyAccess response for each request. Then, the PEP saves the session identifiers for the requests associated with the resources and operations on which access to be granted. Finally, the /token endpoint expresses the resources and operations through a scope and includes it in the response returned to the Client.

As an example (also shown in Figure 5.5) the scope SCOPE could refer to the set of resources and operations {<RES1, READ>, <RES2, WRITE>}, the scope SCOPE1 to {<RES1, READ>}, and the scope SCOPE2 to {<RES2, WRITE>}. Let client C specify SCOPE as scope and AUD as audience in its Access Token request. First, the AS obtains the set of resources and operations corresponding to the scope SCOPE, i.e., {<RES1, READ>, <RES2, WRITE>}. Then, the PEP creates two XACML requests and individually submits them to the UCS. Each request contains C as subject and AUD as audience identifying the resource server. Additionally, one XACML request contains RES1 as resource and READ as action, while the other contains RES2 as resource and WRITE as action. Finally, the UCS evaluates the two requests and returns an access decision for each of them. If, for example, the access decision for <RES1, READ> is Deny and the access decision for <RES2, WRITE> is Permit, then, the AS expresses {<RES2, WRITE>} through the scope SCOPE2 and includes such a scope in its response to the Client.

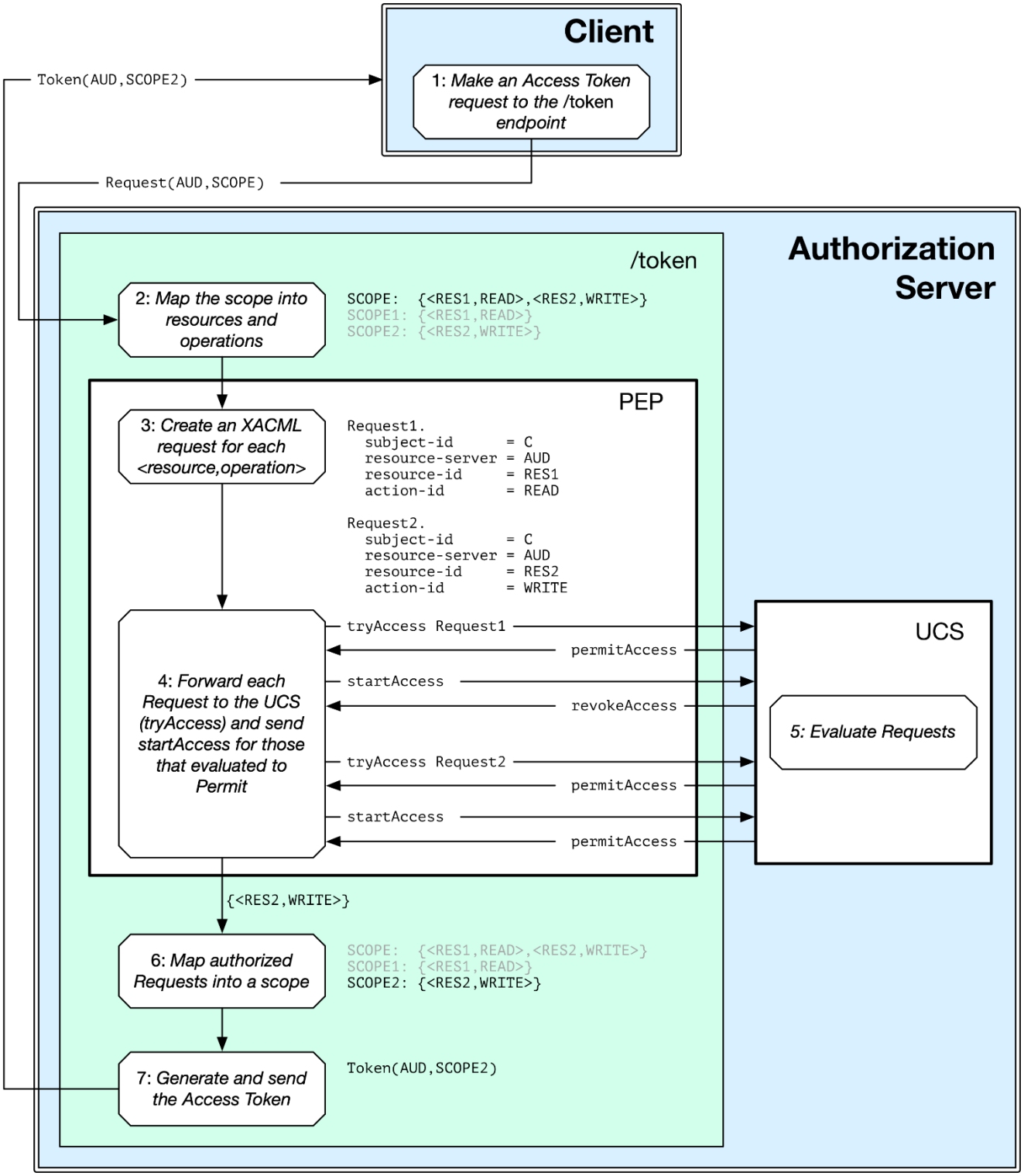


Figure 5.5 - Overview of the UCON framework integrated into the ACE framework.

5.4.2 Access Token Revocation and Notification

The UCON framework provides a continuous monitoring of active usage control sessions and can revoke accesses to resources at the RS according to the mechanisms described in Section 5.3. Its revocation mechanism is fine grained and capable of revoking access to a specific resource for performing a specific operation on it. On the other hand, the ACE framework considers the revocation of an Access Token as a whole. Since a single Access

Token can grant access to more than one resource and operation, this results in the creation of more than one session per Access Token at the UCS. That is, in ACE, the revocation of a Client's access privileges on a resource and operation at an RS implies the revocation of the whole Access Token issued to that Client and to be consumed by that RS. For this reason, an additional customization has been done to correctly integrate the UCON framework into ACE, while preserving a consistent enforcement of access and usage policies.

In the UCON framework, an access is revoked when a policy re-evaluation produces Deny as an access decision. In such a case, the UCS sends a revokeAccess message to the PEP, and the PEP sends an endMessage message to the UCS. Both these messages convey information about the session to be revoked, i.e., the SessionID.

In the proposed integration, the PEP embedded in the AS implements the logic to group together the sessions related to the same Access Token. By doing so, when the PEP receives a revokeAccess message for a specific SessionID, it first finds the Access Token associated with that session, and then it looks for other session identifiers associated with the same Access Token. Finally, it sends an endAccess message to the UCS for each session associated with that Access Token. Moreover, the Access Token identifier, i.e., the token hash of the Access Token, is stored in the TRL resource at the AS (see Section 5.2), whose representation changes over time as valid Access Tokens are revoked or when revoked Access Tokens eventually expire. After that, the additional notification of revoked Access Tokens can take place as per Section 5.2.

In the following, a practical example is given, and the workflow from the Access Token request to the Access Token revocation and related notification is described.

Tenants of a smart home can use a washing machine (the Resource Server) by means of an application installed on their smartphone (Client). The washing machine provides a number of resources, such as "spin cycle" and "high-temp cycle", and the actions defined on such resources might be "start" and "stop". Tenants could be allowed by the administrator to trigger the execution of high-temperature wash cycles, as far as the threshold of daily energy consumption of the overall household is not passed. Consistently, the administrator defines a policy for the tenants and the resource "high-temp cycle", and it adds a condition concerning the attribute "daily energy consumption" in the on-condition of the UCP, specifying that its value must be lower than a certain threshold in order to let tenants start high-temperature wash cycles. The value of the overall daily energy consumption in the household is managed by an AM, which is the smart meter deployed in the smart home.

Within the application, the tenant selects the high-temperature wash cycle and then clicks the start button to begin the washing process. If an Access Token has not already been issued to the tenant and uploaded to the washing machine, or if such an Access Token has ceased to be valid due to expiration or revocation, an Access Token request for the resource "high-temp cycle" and operation "start" is sent to the AS.

The PEP creates an XACML access request and sends it within a tryAccess message to the UCS, which replies with a permitAccess message and creates a session with session identifier SID* and status TRY_ACCESS. Then, the PEP sends a startAccess message to the UCS, which replies with a permitAccess message – if the daily energy consumption is currently lower than the threshold specified in the on-condition of the UCP – and updates the session with the status START_ACCESS. The access is therefore granted, and an Access Token is issued to the Client. Then, the Client uploads the Access Token to the washing machine and establishes a secure communication association with it, according to the specifically used profile of ACE (see Sections 3.8.3 and 5.1). After that, as per the granted access, the tenant is able to trigger the execution of the high-temperature wash cycle by sending a protected request, e.g., a POST request, to the corresponding resource "high-temp cycle" at the washing machine.

From the moment when the UCS updates the status of the session to START_ACCESS, the UCS has been continuously monitoring the mutable attribute "daily energy consumption" since it is in the on-condition of an active session. In case the daily consumption threshold is passed, the Access Token is revoked. That is, the policy

re-evaluation at the UCS returns a Deny access decision because the current daily energy consumption value (retrieved from the smart meter) is higher than the threshold value set within the on-condition of the UCP. Then, the UCS updates the session with the status REVOKE_ACCESS and sends a revokeAccess message containing the session identifier SID* to the PEP. The PEP first finds the Access Token associated with that session and tells the AS that such an Access Token must be revoked. Hence, the AS expunges the Access Token, and the token hash of the revoked Access Token is stored in the TRL of the AS. Then, the AS can send notifications to the registered devices observing the TRL resource and to which the Access Token pertains (see Section 5.2). Finally, the PEP selects all the session identifiers associated with the same Access Token – only SID* in this example – and sends one endAccess message specifying SID* to the UCS, which deletes the related session.

If the tenant attempts to obtain another Access Token within the same day, the AS does not grant a scope allowing access to the resource "high-temp cycle" for performing the previous operation at the washing machine, because the on-condition of the UCP is not satisfied at the time of Access Token request.

Note that, in this example, the occurred revocation does not terminate any ongoing high-temperature wash cycle that is already ongoing as previously started based on the old Access Token. It only prevents from starting a new one, since this would be upon attempting to access the resource "high-temp cycle" without being allowed to. However, if it is critical to stop an ongoing wash cycle after the daily threshold has been reached, other mechanisms should be implemented, such as enforcing on the washing machine a logic that terminates any still ongoing operation previously started by the tenant by virtue of the now expunged Access Token.

5.4.3 Implementation and Experimental Evaluation

An existing Java implementation of ACE from RISE [ACE-DEV] has been extended to include the mechanisms described in Sections 5.2 and 5.3. The resulting software implementation from CNR [ACE-UCON-DEV] allows the AS to use the UCON framework for decision making, as well as the /trl endpoint to communicate to Clients and RSs about pertaining, revoked Access Tokens. This implementation operates as described in Section 5.4, by leveraging the mechanisms introduced in Sections 5.2 and 5.3.

A real testbed consisting of the AS, a Client, and an RS has been deployed to assess both the time performance of the notification mechanism (with and without the Observe extension for CoAP) and the time performance of the operations involving usage control tasks. Since the AS is expected to not be resource constrained, a full-fledged laptop computer was used to run the AS process, while both the Client and RS processes ran on Raspberry Pi devices [RaspberryPi].

The aim of the experiments is to measure various time performance metrics, among which the time required by a Client for obtaining an Access Token and accessing a protected resource at a Resource Server, as well as the time required by the AS for revoking an Access Token. Moreover, all the tests performed follow a common workflow that differs only in the way in which the Client and the RS gain knowledge of the Access Token revocation. The time interval that spans from the time when the AS concludes the revocation to the time when both the Client and the RS gain knowledge of the occurred revocation is also a metric of interest.

The Client can learn about occurred revocations only by polling or observing the TRL resource at the /trl endpoint of the AS. Additionally, it can assume that the Access Token has been revoked when it receives a 4.01 (Unauthorized) response from the RS, after having tried to access a protected resource. On the other hand, in order to learn about occurred revocations, the RS can use the off-the-shelf introspection mechanism provided by ACE, as well as the notification mechanism by polling or observing the TRL resource at the /trl endpoint of the AS. Table 5.6 shows the different tested configurations.

Name	Client	Resource Server

ua-i15	4.01 Unauthorized	introspect every 15s
p15-p15	polling every 15s	polling every 15s
p15-o	polling every 15s	observe
o-p15	observe	polling every 15s
o-o	observe	observe

.

Table 5.6 - Tested configurations

The first phase of the specific tested workflow is reported in Figure 5.7.

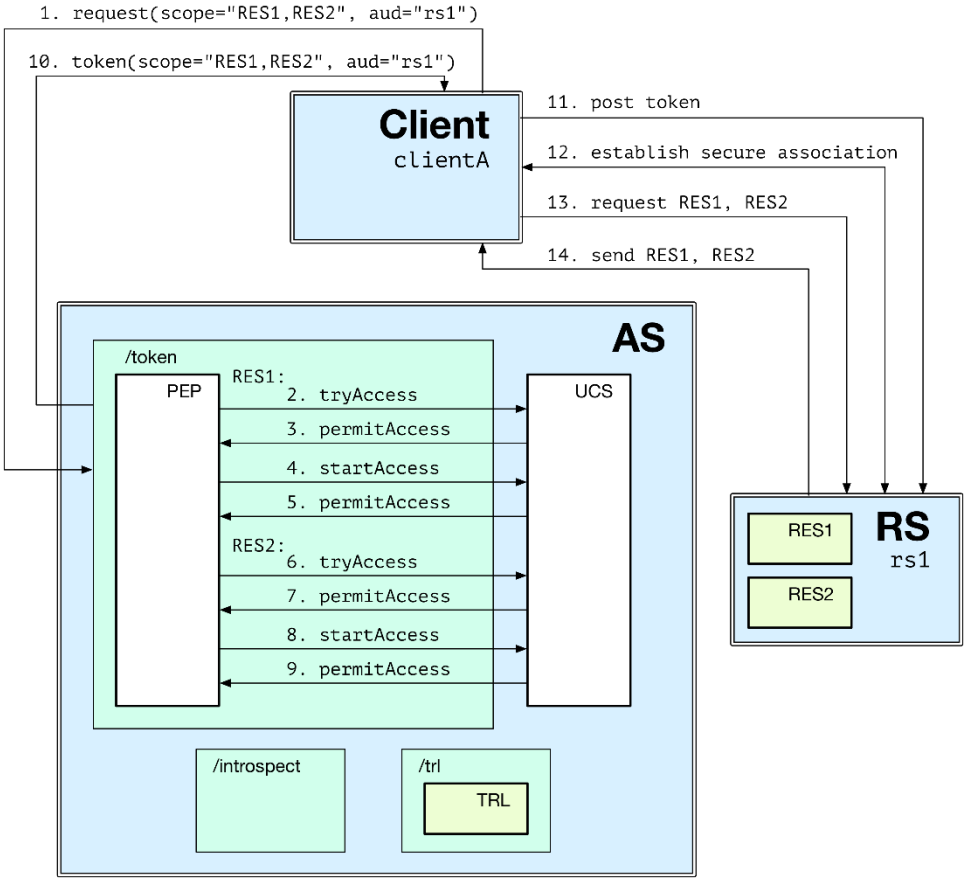


Figure 5.7 - First phase of the tested workflow

In the first phase of the workflow, the Client makes an Access Token request to the AS, specifying "rs1" as audience and "RES1,RES2" as scope. The scope is translated into the set of resources and operations {<RES1, READ>, <RES2, READ]}. Two UCPs (UCP1 and UCP2) installed at the PAP of the UCS grant access to both resources and operations, thus both the tryAccess and startAccess return permitAccess for each request made by the PEP.

The whole requested scope can therefore be granted, and a response containing the Access Token TOKEN1 is sent back to the Client. Then, the Client establishes a secure communication association with the RS "rs1" (in this case, they establish an OSCORE Security Context), and, finally, the Client regularly and alternately sends CoAP GET requests to the RS in order to retrieve the representation of each resource, i.e., RES1 and RES2.

In the second phase of the workflow, the value of an attribute – contained in UCP1 – changes. As soon as the AS notices it, it starts the re-evaluation of UCP1, which results in an access decision Deny. Hence, the AS revokes TOKEN1 and communicates it to the Clients and/or Resource Server according to the configuration currently under test (see Table 5.6).

When the Client learns that TOKEN1 was revoked, the last phase of the workflow is executed. That is, the Client asks the AS for a new Access Token, specifying the same audience and scope used in the first phase of the workflow, as shown in Figure 5.8. In this phase, the evaluation of the on-condition of UCP1 returns an access decision Deny, so the access to RES1 cannot be granted. Therefore, the resulting Access Token TOKEN2 includes the scope "RES2", which allows the Client to access only the resource RES2 at the resource server. The Client establishes a new OSCORE Security Context with "rs1", and the workflow ends when the Client finally retrieves the representation of the resource RES2.

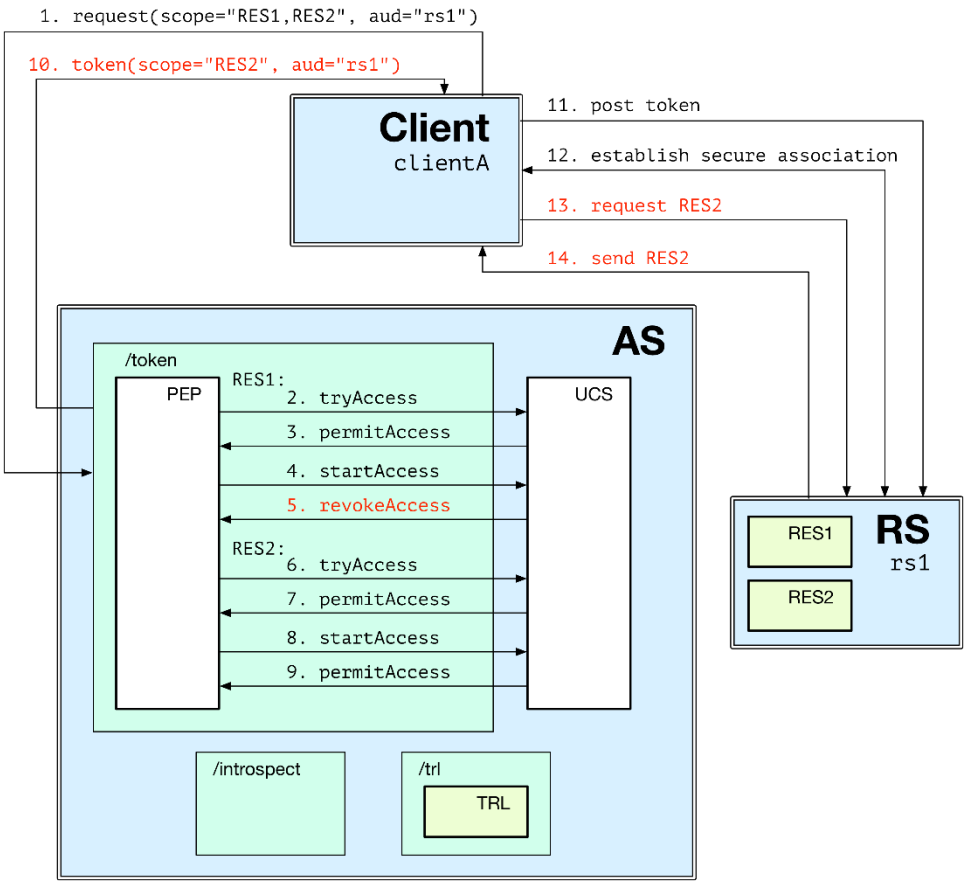


Figure 5.8 - Last phase of the tested workflow

The considered metrics to evaluate the performance of this solution are time intervals, namely:

- **Client Experience Time** (t_{CEx}): it spans from (i) the time when the Client sends an Access Token request to the AS, to (ii) the time when the Client obtains a successful protected response from the RS, following a resource access.
  This time interval measures the time required for a complete execution of the ACE workflow, which consists of the Access Token request and response, the Access Token upload to the RS, the establishment

of a secure communication association with the RS, and the request and response exchanged between Client and RS when accessing the protected resource. From the Client's perspective, this time interval measures the time required to retrieve the representation of a protected resource, i.e., the time to complete the Client's "experience".

- **Revocation Time** (t_{Rev}): it spans from (i) the time when the value of an attribute changes in such a way that triggers the revocation of an Access Token, to (ii) the time when the AS completes the revocation of the Access Token.
  This time interval represents the total time required by the AS for performing the token revocation. It is intended to capture the time spent by the AS to detect that an attribute's value has changed plus the local processing at the AS for completing the token revocation.
- **Inconsistency Time** (t_{Inc}): it spans from (i) the time when the value of an attribute changes, in such a way that triggers the revocation of an Access Token, to (ii) the time when the RS deletes its stored Access Token and terminates the related secure communication association with the Client as a consequence of having gained knowledge of the Access Token revocation.
  This time interval can be split into two parts. The first part is the Revocation Time defined above. The second part is a sort of *information propagation time*. This time interval heavily depends on the mechanism the RS uses to check the token validity, i.e., introspection, polling the /trl endpoint at the AS, or observing the /trl endpoint at the AS.
  This time interval measures an inconsistency, i.e., the time that the Client is still granted the access to a protected resource when it should not be, because the factors (i.e., the attribute) granting such an access have changed thus causing the token revocation.
- **Re-Admission Time** ($t_{Re\text{-}A}$): it spans from (i) the earliest among the time when the RS deletes its stored Access Token as well as the related secure communication association on the one hand, and the time when the Client learns that the Access Token was revoked on the other hand, to (ii) the time when the Client receives a successful, protected response from the RS, following a resource access based on the second Access Token, obtained after the revocation of the first one.
  This time interval captures the time required by the Client to re-access (possibly a subset of) protected resources after either: i) it has learned that an Access Token has been revoked; or ii) the RS has deleted its stored Access Token and the secure communication association with the Client. Consistently with the specific workflow of our experiments, this time interval measures the amount of time that the Client is denied access to RES2, i.e., a protected resource for which it should still have access rights.
  Note that the Client is assumed to be honest, i.e., it does not use an Access Token which it knows to be revoked. If the Client learns before the RS that the Access Token has been revoked, this time interval coincides with the Client Experience Time.

The results of the experiments are shown in Figure 5.9.

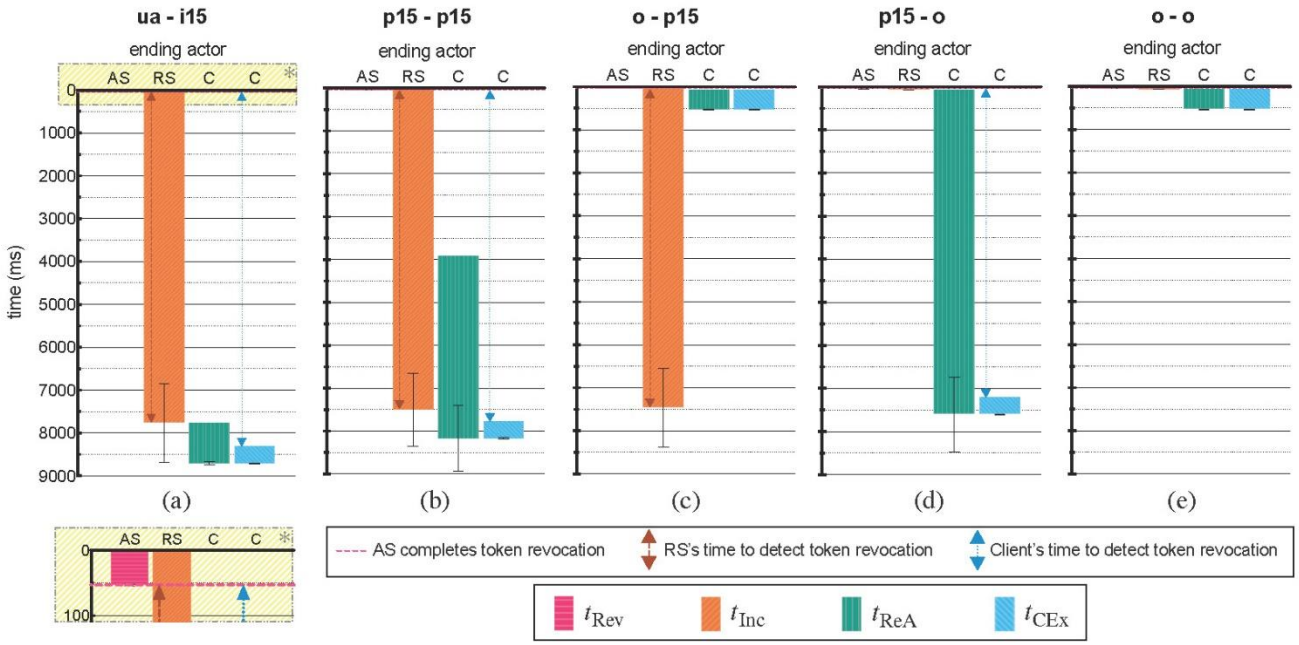


Figure 5.9 - Client Experience Time, Revocation Time, Inconsistency Time, and Re-Admission Time for all the tested configurations. The time t=0 on the y-axis is set to be the time when the value of the attribute changes.

The first configuration (ua-i15) operates as an ACE implementation where no revoked notification mechanism is implemented. Therefore, the RS gains knowledge of the occurred Access Token revocation when it introspects the Access Token at the AS. Subsequently, the Client assumes that the Access Token has been revoked when the RS replies with a 4.01 (Unauthorized) message, following a resource request performed by the Client. The time to notice that the revocation occurred depends on when the RS performs introspection. In these experiments, introspection is performed every 15 seconds, so, on average, the Inconsistency Time is 7.5 seconds plus the Revocation Time, i.e., the time required to the AS to perform the Access Token revocation from the moment when the attribute value has changed. Since this time does not depend on the specifically tested configuration, it is constant, as expected, and its value is about 60 milliseconds. For the same reason, the Client Experience Time is constant, and its value is always lower than 1 second. However, the Client Experience Time depends on the specific UCPs and on the number of UCPs used by the AS when performing the decision-making process before issuing the Access Token for the Client.

Other worth-noting results are obtained considering configurations where either the Client or the RS learns about the Access Token revocation by means of an Observe notification from the AS. In these scenarios, the time to learn is lower than 50 milliseconds, and this minimizes the Inconsistency Time (in configuration p15-o), the Re-Admission Time (in configuration o-p15), or both (in configuration o-o).

6.16 Part 3 – Establishment and Management of Keying Material

This section presents the developed security solutions within the area "Establishment and Management of Keying Material".

Key Provisioning for Group OSCORE using ACE

The solutions and methods presented in this section pertain to the following requirements defined in [D1.2]:

- *Functional Requirements: F-55, F-56, F-57*
- *Non-Functional Requirements: PE-28, PE-29, PE-31*
- *Security Requirements: SE-30, SE-31, SE-35, SE-36, SE-41, SE-44, SE-45*

The solutions and methods presented in this section pertain to the following components defined in [D1.4]:

- *The "Authentication Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Key Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Secure Message Exchange Manager" component of the "Secure Communication Layer" module.*
- *The "Content Distribution Manager" component of the "Secure Communication Layer" module.*

In order to use Group OSCORE [TIL23i] to securely communicate with other CoAP endpoints (see Section 4.1), a node has to explicitly join that group through the associated Group Manager (see Section 4.1.1).

Before doing so, the node has to be explicitly authorized to join the OSCORE group, and has to prove it to the Group Manager. After that, the actual joining procedure can be performed, and the Group Manager can especially provide the joining node with the necessary keying material to communicate in the OSCORE group.

Both tasks can be effectively performed using the ACE framework for authentication and authorization [SEI22] introduced in Section 3.8. In particular, as presented in this section and specified in the standardization proposal [TIL23a], the ACE framework can be used to: i) enforce access control policies at the Group Manager, for candidate members wishing to join an OSCORE group; ii) enforce management of group keying material at the Group Manager, including key provisioning to joining nodes and other operations for current group members.

A Java implementation of such an OSCORE Group Manager from RISE is available at [ACE-DEV], as integrated in the Java implementation from RISE of the overall ACE framework and available for use in the SIFIS-Home project. Also, it has been individually demonstrated through the early focused demo described in Appendix C, and then integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 and the Smart-Home pilot within WP6.

Intuitively, the following mapping occurs between Group OSCORE entities and ACE entities:

- A joining node interested in becoming a member of an OSCORE group acts as ACE client. Thus, it requests an Access Token from an Authorization Server (AS), in order to prove to be authorized to join an OSCORE group with particular roles.
- The Group Manager acts as ACE Resource Server (RS), and is in a secure association with the AS. That is, the Group Manager consumes Access Tokens issued by the AS, and accordingly admits joining nodes to become members of its own OSCORE groups.

Consistently building on the ACE workflow summarized in Section 3.8.2, the following steps occur. The security of the exchange between the joining node and the Group Manager is simply as per the particular security profile of ACE, e.g., [PAL22][GER22], possibly indicated by the AS or pre-configured at the involved parties.

1. The joining node contacts the AS and asks for an Access Token to join one or more OSCORE groups at the Group Manager. When doing so, the joining node indicates the names of the OSCORE groups it intends to join, as well as the role(s) it wishes to have in each of those groups. Possible roles are:
   a. Requester: the node will be interested in sending CoAP requests in the group.
   b. Responder: the node will be interested in sending CoAP responses in the group.
   c. Monitor: the node, while interested in receiving CoAP requests in the group, will never respond to

those and will never send CoAP requests of its own. This corresponds to what is defined as "silent server" in the Group OSCORE protocol.

2. The joining node obtains the Access Token and uploads it at the Group Manager, which validates it and stores it. Contextually, the joining node and the Group Manager establish a secure communication association. As mentioned above, such an establishment as well as the security protocol used between the joining node and the Group Manager depend on the specifically used transport profile of ACE.

3. The joining node sends a Join Request to the Group Manager, targeting the group-membership resource associated with the OSCORE group to join. In the joining request, the joining node specifies:
   a. The name of the exact OSCORE group it wants to join, and the particular role(s) it wishes to take in the group.
   b. Its own public authentication credential, which includes its own public key corresponding to its own private key to use in the group. Examples of formats of authentication credentials include public key certificates (e.g., X.509 [COO08] and C509 certificates [MAT23]), CBOR Web Tokens (CWTs) [JON18], and CWT Claims Sets (CCSs) [JON18].
   c. A Proof-of-Possession (PoP) evidence of its own private key, in order to prove possession of such key to the Group Manager. The PoP input used to compute the PoP evidence consists also of a challenge that both the joining node and the Group Manager can build, using information typically exchanged during the Access Token upload at step 2. If the OSCORE group does not use only the pairwise mode, the PoP evidence is a digital signature that the joining node computes using its private key. If the OSCORE group uses only the pairwise mode, the PoP evidence is a MAC, that the joining node computes using a symmetric key derived from a static-static Diffie-Hellman secret, which is in turn derived from the joining node's and the Group Manager's public authentication credentials.
   d. Optionally, an indication of interest to retrieve the public authentication credentials of the other, current group members.
   e. Optionally, the URL of a local control resource where the Group Manager can send requests to, concerning administrative operations for that group.

4. After validating the Join Request and achieving proof-of-possession of the joining node's private key, the Group Manager authorizes the joining node to access the OSCORE group, and provides it with the following information
   a. The group keying material to use for communicating in the group by using Group OSCORE. This especially includes the OSCORE Master Secret, the OSCORE ID Context used as Group ID, and an OSCORE Sender ID uniquely assigned to the joining node. Furthermore, information and parameters related to the cryptographic algorithms used in the group are also provided.
   b. The Group Manager's public authentication credential, including the public key of the Group Manager.
   c. A PoP evidence of the Group Manager's private key, together with a nonce used as PoP input to compute the PoP evidence. This PoP evidence is computed through the same approach used by the joining node to compute its own PoP evidence included in the Join Request (see step 3c above).
   d. If requested, the public authentication credentials of the other group members, including the corresponding public keys. For each specified public authentication credential, the Group Manager also provides the Sender ID of the corresponding group member.
   e. Optionally, the set of communication policies and security adopted in the group.

5. After validating the Join Response and achieving proof-of-possession of the Group Manager's private key, the joining node can start communicating within the group, protecting its communications with the other group members by using the Group OSCORE security protocol.

Later on, as an active group member, a node can have further secure interaction with the Group Manager, according to the same, dedicated RESTful interface used to carry out the joining operation described above. In particular, a current group member can perform the following operations at the Group Manager, by targeting the same group-membership resource considered during the joining process and associated with the OSCORE group in question, or some of its dedicated sub-resources.

- Request for the current group keying material – A response to this request provides the requester group member with the keying material currently used in the group, similarly to what is indicated in a Join Response (see above). This is useful in case the group member has been repeatedly failing in successfully verifying incoming messages within the group, e.g., due to having missed a group rekeying.
- Request for a new Sender ID – A response to this request provides the requester group member with a new OSCORE Sender ID assigned by the Group Manager. As a more efficient alternative to rejoining the group altogether, this is useful in case the group member has exhausted the values of its Sender Sequence Number space, and requires a new identifier before resuming to communicate in the group.
- Request for the public authentication credential of the Group Manager – A response to this request provides the requester group member with the current, public authentication credential of the Group Manager. This is useful in case the Group Manager has changed its public authentications credentials associated with the group, and the requester group member has missed a previous circulation of such authentication credential.
- Request for the public authentication credentials the current group members – A response to this request provides the requester group member with the public authentication credentials of all the current group members, or of a selected subset of those. This is useful for the requester group member to retrieve the public authentication credentials of group member that have recently joined the group, or of group members that have changed their authentication credential used in the group in question.
- Request for the set of "stale" Sender IDs – A response to this request provides the requester group member with the OSCORE Sender IDs previously associated with other group members and recently relinquished, due to such group members having requested a change of identifier or having left the group. This allows the requester group member to retrieve such identifiers even in case it has missed previous communications from the Group Manager circulating those (e.g., when performing a group rekeying). This ensures that the requester group member can delete the OSCORE Recipient Contexts and authentication credentials associated to such stale identifiers, which in turn ensures the ability to confidently assert whether the sender of a received message is currently a member of the group.
- Request for the current communication policies in the OSCORE group – A response to this request provides the requester group member with the current set of communication and security policies used in the group.
- Request for the current version of the group keying material – A response to this request provides the requester group member with the current version of the group keying material used in the group, as an unsigned integer that get incremented by 1 every time that the group is rekeyed. This efficiently allows a group member to determine whether it has missed the execution of a group rekeying, and consequently to follow-up with a separate request to the Group Manager for retrieving the current group keying material.
- Request for the current status of the OSCORE group – A response to this request provides the requester group member with the current status of the group, i.e., active or inactive. If the group status is set to inactive, current group members should refrain from communicating, while new members will not be allowed to join.
- Provide the Group Manager with a new public authentication credential – This allows the requester group member to provide the Group Manager with its own, new public authentication credential, including its

new, own public key. This has to be provided with a new PoP evidence, based on the same approaches used for the PoP evidence included in a Join Request (see above). If the Group Manager successfully achieves proof-of-possession of the group member's new private key, the Group Manager replaces the stored, old group member's public authentication credential with the new one.

- Leave the OSCORE group – This allows the requester group member to notify the Group Manager that it is leaving the group, thus facilitating the Group Manager in follow-up maintenance operations such as the execution of a group rekeying.

During the group lifetime, the Group Manager can forcefully evict group members, as well as distribute new keying material (rekeying) to the current group members. The proposal at [TIL23a] considers a basic rekeying approach, where the Group Manager provides the new group keying material to each node individually, with one-to-one messages targeting the control resource of each group member, as indicated at joining time. Alternatively, group members may observe [HAR15] the group-membership resource at the Group Manager, to automatically get notifications conveying the latest updated group keying material.

The Group Manager may, however, rely on more efficient approaches available in the literature in order to distribute new keying material in the group, such as [WAL99][WON00][DIN11][DIN13][TIL16][TIL20].

Regardless of the specific approach used to rekey the group, the Group Manager contextually informs the current group members about nodes that have left the group (i.e., about their Sender IDs), thus allowing them to purge related information like associated OSCORE Recipient Contexts and stored public authentication credentials. This in turn makes it possible to preserve the capability for current group members to confidently assert whether the sender of a received message is currently a member of the group.

Notably, additional design and development activities have also been ongoing within the SIFIS-Home project on defining two additional services that are required to cover the full lifecycle of an OSCORE group. Due to time constraints and prioritization choices, these results have still not reached a maturity level that could enable their integration in the SIFIS-Home solution. For information, those activities are shortly summarized below.

First, as also specified in the standardization proposal at [TIL23b], authorized entities acting as Administrators can act as further ACE clients, and securely interact with the Group Manager in order to create, configure, and delete OSCORE groups at that Group Manager. Second, as also specified in the standardization proposal at [TIL23c], deployed devices can rely on the standard CoRE Resource Directory [AMS22], in order to discover an OSCORE group of interest, and especially which Group Manager they should contact in order to join it as new group members, by sending a Join Request to the Group Manager's group-membership resource.

6.2

EDHOC – Key Establishment for OSCORE

The solutions and methods presented in this section pertain to the following requirements defined in [D1.2]:

- *Non-Functional Requirements: PE-28, PE-29, PE-36*
- *Security Requirements: SE-30, SE-31, SE-35, SE-43, SE-47*

The solutions and methods presented in this section pertain to the following components defined in [D1.4]:

- *The "Key Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Secure Message Exchange Manager" component of the "Secure Communication Layer" module.*
- *The "Content Distribution Manager" component of the "Secure Communication Layer" module.*

Ephemeral Diffie-Hellman over COSE (EDHOC) [SEL23a] is a very compact and lightweight authenticated protocol for performing a security handshake and establishing a cryptographic secret between two peers. EDHOC messages can be transported over CoAP, and the main use case is the establishment of an OSCORE Security Context for protecting communications with OSCORE [SEL19] (see Section 3.7).

EDHOC provides a number of high-level security properties, i.e., mutual authentication of the two peers, forward secrecy of the established security material, identity protection, and negotiation of the crypto algorithms to use during its execution (i.e., a cipher suite). This can be achieved through sustainable low power operations, thanks to the protocol design targeting a small overhead and associated processing.

To this end, EDHOC relies on building blocks which in turn are lightweight IETF standards. These include CBOR [BOR20] for message and data encoding (see Section 3.5) and COSE [SCH22a][SCH22b] for cryptographic operations (see Section 3.5). While EDHOC is not bound to a particular protocol for message transport, the CoAP protocol [SHE14] (see Section 3.1) is a practically convenient choice for transporting EDHOC messages. Especially for devices already relying on a communication stack based on CoAP and OSCORE, this makes it possible to keep the additional code size due to EDHOC very low.

As to its core key establishment process, EDHOC makes use of known protocol constructions, such as the SIGMA protocol [KRA03] as well as Extract-and-Expand [RFC5869]. In such a context, COSE further provides crypto agility and enables the use of future algorithms and credential types targeting the IoT.

In particular, EDHOC incarnates the SIGMA-I MAC-then-Sign variant of SIGMA. That is, the identity of the Initiator peer that starts the protocol by sending the first EDHOC message is protected against active attackers, while the identity of the other, Responder peer is protected against passive attackers. Also, except for the first unprotected EDHOC message, the following messages exchanged in an EDHOC execution convey encrypted content. The original plaintext is integrity protected by means of a Message Authentication Code (MAC), which in turn is signed when specifically using an authentication mode that relies on digital signature.

The ultimate goal of EDHOC is for two peers to mutually authenticate and to agree on a session key PRK_out with good security properties, especially forward secrecy. That is, compromising the private authentication key of a peer or an established session key does not compromise any previously established session key. Such an establishment is accompanied by the two peers mutually authenticating and achieving key confirmation, i.e., each peer asserts that the other peer has also derived the same session key. Also, the two peers achieve proof-of-possession, i.e., each peer confirms to the other peer the possession of its own private authentication key. From the EDHOC session key, further keying material can be exported, such as an OSCORE Security Context.

An EDHOC execution consists of three mandatory messages, i.e., message_1, message_2, and message_3. The Initiator peer starts the protocol by sending EDHOC message_1, after which the other, Responder peer replies with message_2, followed by the Initiator sending message_3. An optional message_4 can be sent by the Responder after receiving message_3, to ensure that the Initiator achieves key confirmation in scenarios where the Responder never sends protected application messages to the Initiator. EDHOC also provides an error message, which is always fatal and can be sent to abort the protocol execution at any point in time.

Throughout an EDHOC execution, the two peers exchange the following information.

Connection identifiers. During an EDHOC execution a peer refers to a corresponding EDHOC session, chooses its own EDHOC Connection Identifier associated with such session, and offers it to the other peer. When receiving an EDHOC message different from message_1, each peer uses its own connection identifier to retrieve the ongoing EDHOC session and its status. Connection identifiers are intrinsically byte strings. When using a

transport that does not provide full correlation between a message and the immediately previous one, connection identifiers can be prepended to EDHOC messages, thus enabling the recipient peer to retrieve the right EDHOC session. In order to limit communication overhead, a particular set of connection identifiers are encoded as CBOR integers when sent on the wire, which limits their message overhead to only 1 byte.

Cipher suite. A specific set of algorithms is used for cryptographic operations such as authenticated encryption, digital signature and key derivation. Such a set of algorithms is defined as a cipher suite. During the first phase of the protocol, the two EDHOC peers securely negotiate the cipher suite to use, ensuring that they run EDHOC using the cipher suite that they both support and that the Initiator prefers.

Public, ephemeral Diffie-Hellman keys. Each peer generates a pair of ephemeral Diffie-Hellman keys, and provides the public one to the other peer. Each peer relies on its own ephemeral private key and the other peer's ephemeral public key in order to derive a common Diffie-Hellman shared secret G^{XY}. The shared secret takes part to the key derivation of the final EDHOC session key PRK_out, hence contributing to its forward secrecy.

Authentication method. As mentioned above, EDHOC also ensures mutual authentication of the two peers. To this end, an EDHOC peer can rely on one of two possible methods, with no need for both peers to use the same method during an EDHOC execution. The first method relies on public keys as signing keys, thus resulting in a digital signature as authentication evidence. The second method relies on a secret derived from a private static Diffie-Hellman key, which in turn is used to compute a MAC as authentication evidence. Clearly, the latter method yields EDHOC messages that are smaller in size. An "EDHOC authentication method" identifies the specific combination of individual authentication methods used by the two peers in an EDHOC execution.

Authentication credentials. EDHOC admits only authentication credentials based on asymmetric public keys. In turn, these can have different formats (e.g., CBOR Web Tokens (CWTs) / CWT Claims Sets (CCSs) [JON18], X.509 certificates [COO08] and CBOR encoded X.509 (C509) certificates [MAT23]). Also, the associated public key can be for a signature algorithm (e.g., EdDSA), or a key agreement algorithm (e.g., X25519). Furthermore, during an EDHOC execution, the two peers can exchange the actual authentication credentials by value, or instead provide each other with their references (e.g., binary identifiers or hashes) in case credentials are pre-provided or retrievable from other sources. Besides being used to achieve peer authentication, the authentication credentials also take part to the derivation of the final EDHOC session key PRK_out, thus protecting against identity misbinding attacks.

External Authorization Data (EAD). Each EDHOC message can optionally specify an EAD field, each including an EAD item identified by a numeric label and optionally conveying data. These items are used to transport information from external security applications, whose execution can be efficiently embedded within an EDHOC execution. Examples of such applications include uploading access control credentials, network authentication, and public certificate enrollment. EAD items can also be used to pad EDHOC messages.

Figure 6.1 shows an EDHOC execution, during which the two peers exchange the information elements above.

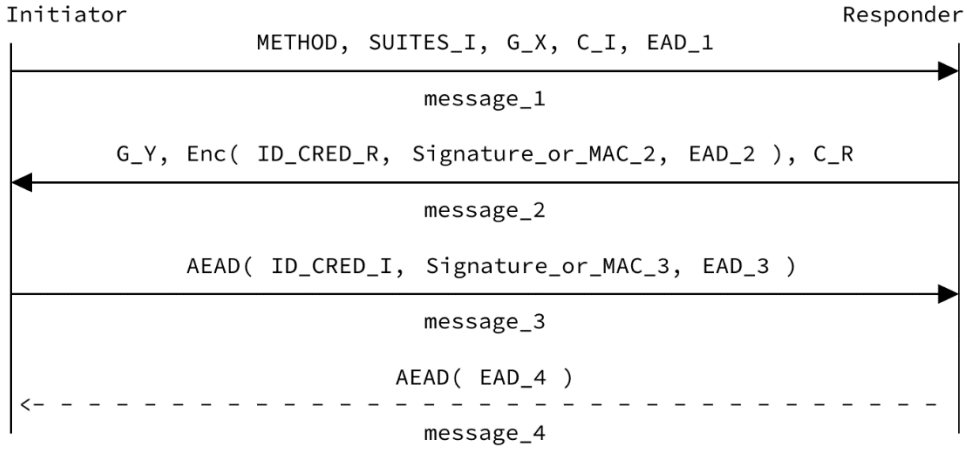


Figure 6.1 - Execution of the EDHOC protocol, including the optional fourth message

The Initiator specifies in message_1: the EDHOC authentication method to use during the EDHOC execution; the cipher suites that is supports, highlighting the preferred one; its ephemeral public Diffie-Hellman key G_X; its own connection identifier C_I, and, optionally, EAD items in the EAD_1 field. This first EDHOC message is not protected.

The Responder specifies in message_2: its ephemeral public Diffie-Hellman key G_Y, its own connection identifier C_R, and an encrypted blob. The latter is produced through binary additive stream cipher encryption and includes: the identifier ID_CRED_R of the Responder's authentication credential; a signature or a MAC as authentication evidence for the Responder (see above), and, optionally EAD items in the EAD_2 field.

In message_3 the Initiator specifies an encrypted blob, which is produced through Authenticated Encryption with Additional Data (AEAD) and includes: the identifier ID_CRED_I of the Initiator's authentication credential; a signature or a MAC as authentication evidence for the Initiator (see above), and, optionally EAD items in the EAD_3 field.

In the optional message_4, the Responder specifies an encrypted blob, which is produced through AEAD and optionally includes EAD items in the EAD_4 field.

Throughout the EDHOC execution, the keying material used to protect message_2, message_3 and message_4 is incrementally derived by taking as input: the Diffie-Hellman shared secret G^{XY}; keying material derived from the previous steps; and transcript hashes computed from EDHOC messages exchanged at previous steps and from the peers' authentication credentials. This further ensures cryptographic binding across the different messages of the same EDHOC execution. The keying material available at the final stage is used as input to derive the EDHOC session key PRK_out, from which further material can in turn be securely derived.

The Initiator authenticates the Responder and achieves proof-of-possession for the Responder's private key when receiving and successfully verifying message_2. The Responder authenticates the Initiator and achieves proof-of-possession for the Initiator's private key when receiving and successfully verifying message_3.

The Responder achieves key confirmation when receiving and successfully verifying message_3. The initiator achieves key confirmation when receiving from the Responder and successfully verifying either the optional message_4, or a first application message protected with application keying material derived from the EDHOC session key.

Examples of message sizes for EDHOC are given in Section 1.3 of [SEL23a]. More details on the design, features, properties and current status of the EDHOC protocol are provided in [VUC22].

If a transport protocol based on the Client-Server paradigm is used, it is typical for a client peer to act as Initiator, although the reverse approach where the server peer acts as initiator is also supported. Section 6.3.1 specifically considers the use of CoAP to transport EDHOC messages and describes an optimized EDHOC execution for the typical workflow where a CoAP client acts as Initiator.

During its development, EDHOC has received extensive formal analysis from cryptographic teams [NOR20][COT22][JAC23][GUE22], which significantly contributed to improve the protocol as to its security properties, its message format and encoding, and the alignment of key derivation with security best practices.

An implementation of the EDHOC protocol from RISE for the Californium library [CALIFORNIUM] from the Eclipse Foundation is available at [EDHOC-DEV] for use in the SIFIS-Home project. Also, it has been individually demonstrated through the early focused demo described in Appendix D, and then integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 and the Smart-Home pilot within WP6.

As mentioned above, EDHOC has as a main use case the establishment of an OSCORE Security Context, through an execution performed over CoAP. Later at some point, the two peers may need to update their OSCORE Security Context. For example, the keying material becomes too old or unsafe to use due to passed cryptographic limits, or one peer has exhausted the values for its Sender Sequence Number.

The two peers can certainly perform a new execution of EDHOC, with the consequent establishment of a totally independent, new OSCORE Security Context. However, it would be more convenient for the two peers to rely on a dedicated, more efficient and lightweight procedure that simply updates the current OSCORE Security Context.

To this end, additional design and development activities have also been ongoing within the SIFIS-Home project for defining a lightweight protocol to perform key update for OSCORE, namely KUDOS [HÖG23]. Due to time constraints and prioritization choices, these results have still not reached a maturity level that could enable their integration in the SIFIS-Home solution.

For information, the KUDOS protocol is lightweight and agnostic of the originally used establishment method, can be initiated by either the CoAP client or the CoAP server, completes in only one round trip, preserves forward secrecy, and allows the two peers to preserve their ongoing CoAP observations beyond the update of their OSCORE Security Context.

6.2.1 Using EDHOC with CoAP and OSCORE

Even though EDHOC is independent of the used transfer and transport protocol, EDHOC messages can be specifically transported as payload of CoAP messages. Along the same lines, the cryptographic secret established through EDHOC can have broad, general applicability. However, the main use case for EDHOC is the ultimate establishment of an OSCORE Security Context (see Section 3.7.1), as derived from the secret established through an EDHOC execution between the two peers. In short, EDHOC is especially convenient and expected to be used over CoAP in order to establish keying material for OSCORE.

As per its basic flow, an EDHOC execution requires the two peers to exchange at least three EDHOC messages, with the possible addition of an optional, fourth message. With reference to the CoAP protocol, the following assumes that a CoAP client takes the role of EDHOC Initiator (thus sending the first EDHOC message) and a CoAP server to take the role of EDHOC Responder, according to the EDHOC forward message flow.

As shown in Figure 6.2, the forward message flow consists in the CoAP client sending a request to an EDHOC resource hosted at the CoAP server, specifying EDHOC message_1 as payload. Then, the CoAP server replies with a response, specifying EDHOC message_2 as payload. Finally, the CoAP client sends one more request to the same EDHOC resource at the CoAP server, specifying EDHOC message_3 as payload. The figure also shows the CoAP server replying with a response whose payload conveys the optional EDHOC message_4. After that, EDHOC is completed, the client and server have authenticated one another, and they agree on a cryptographic secret, from which they derive an OSCORE Security Context. Then, the client and server can exchange further CoAP messages protected with OSCORE.

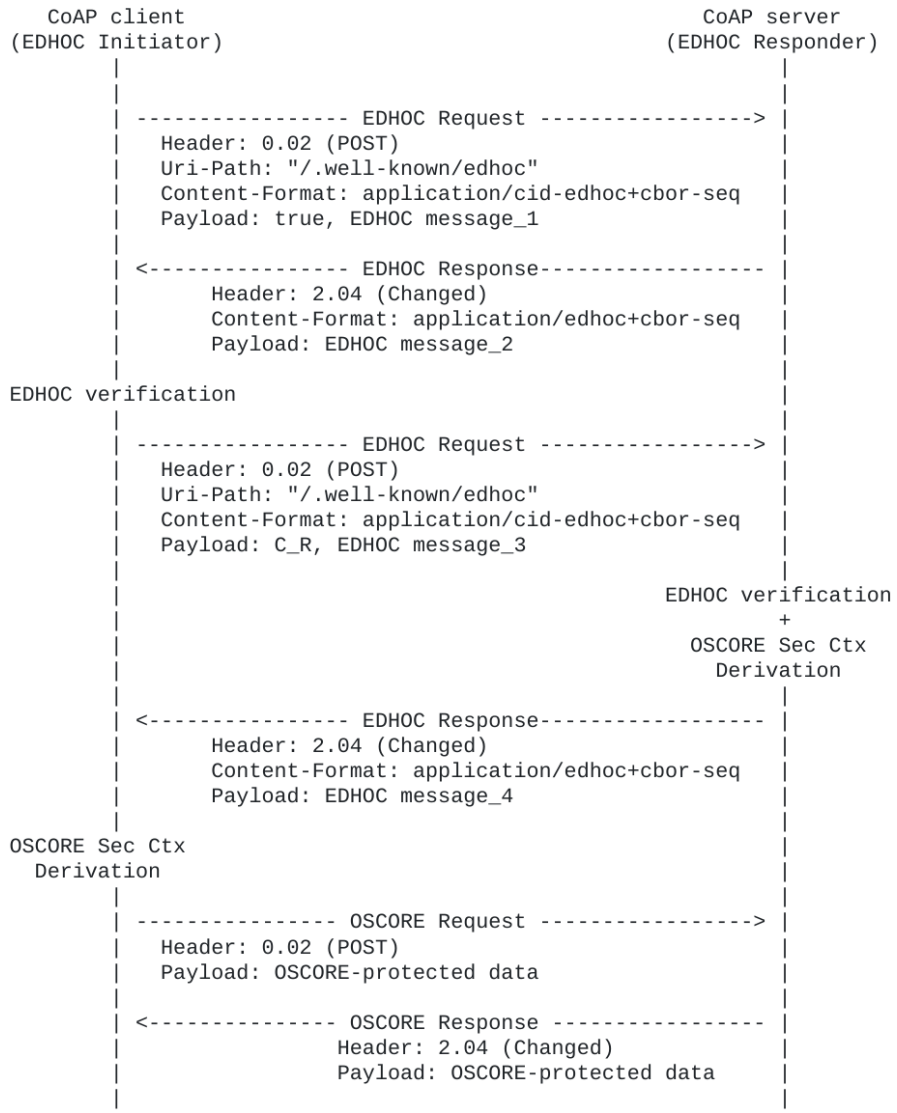


Figure 6.2 - Execution of the EDHOC protocol over CoAP (Forward Message Flow)

Within the SIFIS-Home project, work has also been carried out specifically on specializing and optimizing the

use of EDHOC with CoAP and as a means to establish an OSCORE Security Context. This activity is also a current IETF standardization proposal [PAL23b], and can be broken down into the three following contributions.

Optimized EDHOC execution. The message flow discussed above and shown in Figure 6.2 is eligible for a major optimization. That is, the CoAP client has all the information to derive the OSCORE Security Context *already after receiving EDHOC message_2*, i.e., before sending EDHOC message_3 to the CoAP server.

This means that, at that point in time, the client is already able to produce not only the following EDHOC message_3, but also the subsequent OSCORE-protected application request. Clearly, it would be ideal to combine those two requests into a single one to be sent on the wire.

This outcome is practically achieved as defined in [PAL23b] and shown below in Figure 6.3. Intuitively, after having processed EDHOC message_2 and established the OSCORE Security Context, the CoAP client sends a single, combined EDHOC + OSCORE request to the CoAP server. The combined request does not target the same EDHOC resource as the previous request conveying EDHOC message_1, but rather the resource where the client intends to send its OSCORE-protected application request. From a semantic point-of-view, such a combined request effectively includes two typically independent requests, i.e., a request conveying EDHOC message_3 and a further OSCORE-protected request intended to an actual application resource.

In particular, the EDHOC + OSCORE request includes a newly defined “EDHOC” CoAP option, which signals to the CoAP server that the message is indeed a combined request, hence that its payload conveys both EDHOC message_3 and OSCORE-protected application data. As its only intent is simple signaling equivalently to a flag, the option content is always empty, and results in only 1 byte of message overhead. Except for EDHOC message_3 itself, the client protects this combined request using the established OSCORE Security Context.

Specifically, after having received and correctly processed EDHOC message_2, the client proceeds as follows. First, it prepares EDHOC message_3 and derives the new OSCORE Security Context from the cryptographic secret resulting from the EDHOC execution. Then the client prepares the actual CoAP application request that it intends to send to the server, and encrypts the request with the newly established Security Context (see Section 3.7.2). After that, the client considers the OSCORE-protected request, and prepends the current request payload (i.e., the OSCORE ciphertext) with the self-delimiting EDHOC message_3. The resulting request payload consists of EDHOC message_3 concatenated with the original OSCORE ciphertext. Note that, unlike in the basic EDHOC execution flow, EDHOC message_3 is not prepended by the EDHOC connection identifier C_R, which allows to further reduce the communication overhead. This is possible thanks to the fact that C_R is the OSCORE Sender ID of the client, which is already specified as value of the ‘kid’ field in the OSCORE Option of the request (see Section 3.7.2). Finally, the client adds the EDHOC Option to the request, thus signaling that the message is specifically an EDHOC + OSCORE request, and sends the request to the server.

When receiving the EDHOC + OSCORE request, the server notices the signaling EDHOC Option and proceeds as follows. First, the server extracts EDHOC message_3 from the request payload. Also, it considers the value of the ‘kid’ field of the OSCORE option as the EDHOC connection identifier C_R for retrieving the correct EDHOC session with the client. Then, the server performs the same operations as if it had received a stand-alone EDHOC message_3 from the client, as related to the EDHOC session retrieved through the connection identifier C_R. After completing the EDHOC execution, the server derives the OSCORE Security Context shared with the client from the cryptographic secret resulting from the EDHOC execution. Then, the server removes the EDHOC Option and EDHOC message_3 from the received request. The result is an OSCORE-protected data request, that the server decrypts by using the newly derived OSCORE Security Context, as per the typical OSCORE processing. The decrypted request can finally be delivered to the server application for further processing, which will result in an OSCORE-protected response sent to the client as a reply.

By combining EDHOC message_3 with the first protected application request into a single CoAP message sent on the wire, this optimization allows for a minimum number of round trips necessary to setup the OSCORE Security Context and complete an OSCORE transaction, e.g., when an IoT device is deployed in a network and configured for the first time. This optimized workflow can be used only if the CoAP client acts as EDHOC

initiator (i.e., in the EDHOC forward message flow), which is however the typical and default case. Also, it does not admit the use of the optional EDHOC message_4, which becomes however not necessary since it is effectively replaced by the first OSCORE-protected response sent by the server.

The optimized EDHOC workflow discussed above has been implemented by RISE and integrated in its Java implementation of EDHOC [EDHOC-DEV] for the Californium library [CALIFORNIUM] from the Eclipse Foundation, as available for use in the SIFIS-Home project. Also, it has been individually demonstrated through the early focused demo described in Appendix D, and then integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 and the Smart-Home pilot within WP6.

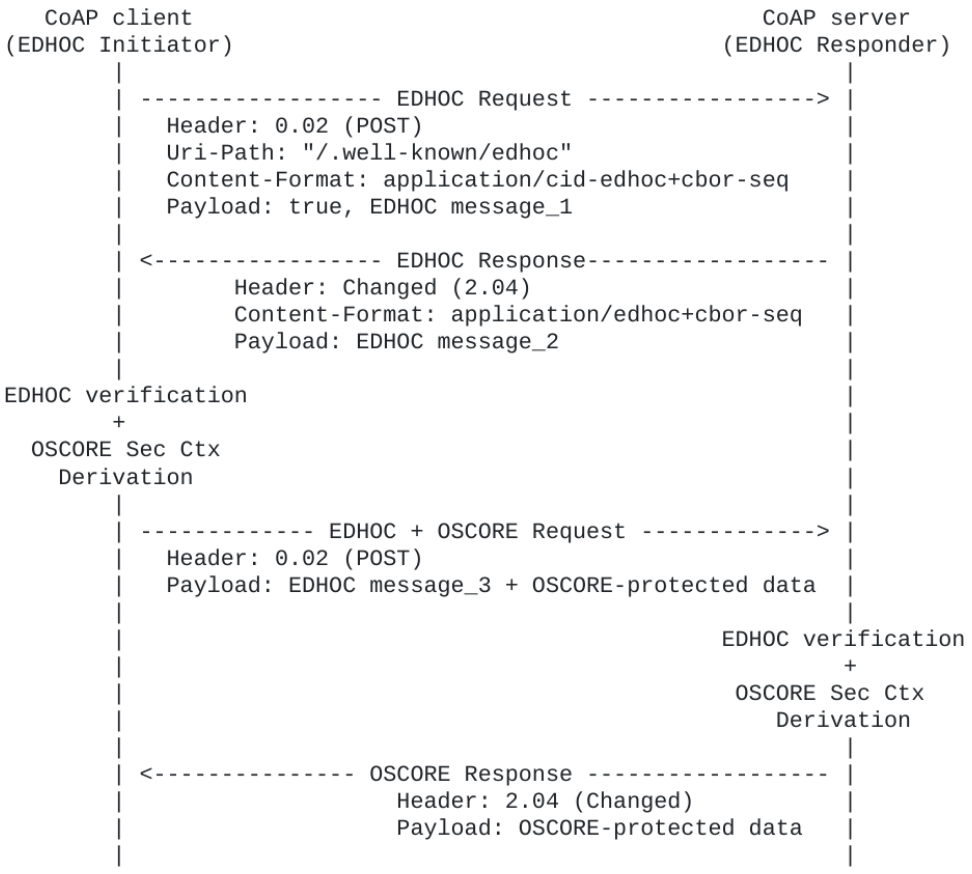


Figure 6.3 - Optimized execution of the EDHOC protocol over CoAP

Handling of EDHOC identifiers as OSCORE identifiers. Using EDHOC to establish an OSCORE Security Context involves two types of identifiers. On the one hand, EDHOC relies on connection identifiers, that are chosen during the EDHOC execution and that the two peers can use to relate one EDHOC message to other messages sent during the same EDHOC execution. On the other hand, OSCORE relies on the Sender/Recipient IDs associated with the two peers sharing an OSCORE Security Context (see Section 3.7.1).

Since EDHOC connection identifiers are intrinsically byte strings, they are straightforwardly converted to and from OSCORE Sender/Recipient IDs. The latter ones are used by the two peers when protecting their communication using the OSCORE Security Context established thanks to EDHOC.

At the same time, additional care must be put by the two peers when running EDHOC with the ultimate goal of deriving an OSCORE Security Context. In particular, the following applies during an EDHOC execution, both when using the basic EDHOC flow or the optimized flow presented above.

- The Initiator must choose its EDHOC connection identifier C_I such that: i) it is currently not used as connection identifier for this peer in any other EDHOC session; and ii) it is currently not used as Recipient ID in an OSCORE Security Context of this peer where the ID Context is not present.

- The Responder must choose its EDHOC connection identifier C_R such that: i) it is different from the connection identifier C_I used in the present EDHOC session by the other peer; ii) it is currently not used as connection identifier for this peer in any other EDHOC session; and iii) it is currently not used as Recipient ID in an OSCORE Security Context of this peer where the ID Context is not present.

- Upon receiving EDHOC message_2, the Initiator must assert whether the conveyed connection identifier C_R is equal to the connection identifier C_I specified in the previous EDHOC message_1 of the same EDHOC execution. In such a case, the Initiator must abort the EDHOC execution.

Web linking to facilitate discovery of EDHOC resources. As explained above, an EDHOC execution occurring over CoAP involves the interaction with an EDHOC resource at a CoAP server. One or multiple EDHOC resources at the same server can be associated with an EDHOC "application profile". This includes a number of information elements describing how EDHOC can be run with that server, when interacting with any of its associated EDHOC resources.

At the same time, there are means for a CoAP client to discover the link to an EDHOC resource, e.g., directly from the server by accessing the resource /.well-known/core at the server, or indirectly by using the CoRE Resource Directory [AMS22]. In either case, the client receives a discovery response in CoRE link-format [SHE12], specifying, among others, the links to the EDHOC resources at the server.

This contribution defines a number of parameters, each of which corresponds to different information elements possibly present in an EDHOC application profile. In particular, these parameters can be used as target attributes accompanying a discovered link to an EDHOC resource, consistently with the CoRE link-format.

The following compiles a list of the defined target attributes. Value for the target attributes are taken from dedicated IANA registries related to EDHOC and COSE.

- ed-i – If present, it specifies that the server supports the EDHOC Initiator role. It takes no value.
- ed-r – If present, it specifies that the server supports the EDHOC Responder role. It takes no value.
- ed-method – Each occurrence specifies a supported EDHOC authentication method.
- ed-csuite – Each occurrence specifies a supported EDHOC cipher suite.
- ed-cred-t – Each occurrence specifies a supported type of authentication credential.
- ed-idcred-t – Each occurrence specifies a supported type of authentication credential identifier.
- ed-ead – Each occurrence specifies the label of a supported EAD item.
- ed-comb-req – If present, it specifies that the server supports the EDHOC + OSCORE request. It takes no value.

Thus, a CoAP client that discovers the link to an EDHOC resource can at the same time learn about the application profile associated with that resource, as described by the specified target attributes. This in turn provides the client with an early knowledge of how to run EDHOC with the server, which also prevents potential negotiations or trial-and-error exchanges to occur during the EDHOC execution, with evident benefits in terms of performance and completion time.

7 Conclusion

This document is the third and final deliverable from WP3 "Network and System Security", and has provided the final description of the network & system security solutions designed and developed in the SIFIS-Home project. In particular, this document obsoletes the previous deliverable D3.2 “Preliminary report on Network and System Security Solutions” from WP3, hence acting as a self-standing description of the network and system security solutions developed during the project and integrated into the SIFIS-Home solution.

The presented security solutions have been grouped under the three following activity areas: i) Secure and Robust (Group) Communication; ii) Access and Usage Control for Server Resources; and iii) Establishment and Management of Keying Material. These are applicable especially – but not only – to the use cases and IoT-based Smart Home environment considered in this project. Also, they are designed to be scalable, efficient and effective, thus limiting the impact on performance of the networks and applications involving also IoT devices.

The security solutions have been explicitly related to the pertaining requirements documented in deliverable D1.2 "Final Architecture Requirements Report", as well as to the SIFIS-Home architecture and its components documented in deliverable D1.4 "Final Component, Architecture, and Intercommunication Design".

Solutions presented in this document have been first implemented and individually demonstrated through early, focused demos. Such individual implementations are available as Open-Source Software.

Through continuous refinements, the security solutions presented in this document have been integrated in the SIFIS-Home solution, as part of the SIFIS-Home testbed within WP5 “Integration, Testing and Demonstration” and in the Smart-Home pilot within WP6 “Smart Home Pilot Use Case”. This relied on integration-oriented components of the SIFIS-Home architecture, and on the process for continuous integration/deployment of Software used in the project. The integrated implementation of the security solutions is also available as Open-Source Software in the project Github organization. Further details on the integration of the security solutions are provided in the deliverable D5.4 “Final version of the SIFIS-Home Security Architecture Implementation”.

Finally, the work and results from WP3 have substantially contributed to dissemination activities in WP7 "Dissemination, Standardization and Exploitation", with reference to academic publications in international venues and technical seminars, and to standardization activities in the Working Groups CoRE, ACE, LAKE and SCHC of the renowned international body Internet Engineering Task Force (IETF).

References

[ACE-DEV] ACE framework Java Implementation. Online: https://bitbucket.org/marco-tiloca-sics/ace-java/src/master/ (fetched in May 2023)

[ACE-UCON-DEV] Usage Control System (UCS) Java implementation for the ACE framework. Online: https://bitbucket.org/marco-rasori-iit/ace-java/src/ucs/ (fetched in May 2023)

[AMS22] C. Amsüss, Z. Shelby, M. Koster, C. Bormann, and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176 (Proposed Standard), Internet Engineering Task Force, RFC Editor, April 2022.

[AMS23] C. Amsüss and M. Tiloca, "Cacheable OSCORE", Internet-Draft draft-amsuess-core-cachable-oscore-06 (work in progress), IETF Secretariat, January 2023.

[BAL21] WSO2 Balana implementation: https://github.com/wso2/balana (fetched in May 2023)

[BAR18] E. Barker, L. Chen, A. Roginsky, A. Vassilev and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography - NIST Special Publication 800-56A, Revision 3", April 2018. Online: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf

[BOR14] C. Bormann, M. Ersue and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228 (Informational), Internet Engineering Task Force, RFC Editor, May 2014.

[BOR16] C. Bormann and Z. Shelby, "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959 (Proposed Standard), Internet Engineering Task Force, RFC Editor, August 2016.

[BOR18] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323 (Proposed Standard), Internet Engineering Task Force, RFC Editor, February 2018.

[BOR20] C. Bormann and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 8949 (Internet Standard), Internet Engineering Task Force, RFC Editor, December 2020.

[BRA17] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259 (Internet Standard), Internet Engineering Task Force, RFC Editor, December 2017.

[CALIFORNIUM] Eclipse/Californium: CoAP/DTLS Java Implementation. Online: https://github.com/eclipse/californium/ (fetched in May 2023)

[CAS17] A. P. Castellani, S. Loreto, A. Rahman, T. Fossati and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075 (Proposed Standard), Internet Engineering Task Force, RFC Editor, February 2017,

[Contiki-NG] The Contiki-NG Operating System. Online: https://www.contiki-ng.org/ (fetched in May 2023)

[COO08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and T. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280 (Proposed Standard), Internet Engineering Task Force, RFC Editor, May 2008.

[COT22] B. Cottier and D. Pointcheval, "Security Analysis of the EDHOC protocol", September 2022, https://arxiv.org/abs/2209.03599 (fetched in May 2023)

[D1.2] SIFIS-Home Deliverable D1.2 "Final Architecture Requirements Report", September 2021.

[D1.4] SIFIS-Home Deliverable D1.4 "Final Component, Architecture, and Intercommunication Design", September 2022.

[D3.2] SIFIS-Home Deliverable D3.2 "Preliminary report on Network and System Security Solutions", March 2022.

[D5.4] SIFIS-Home Deliverable D5.4 "Final version of the SIFIS-Home Security Architecture Implementation", June 2023.

[DEG11] J. P. Degabriele, A. Lehmann, K. Paterson, N. Smart and M. Strefler, "On the Joint Security of Encryption and Signature in EMV", Cryptology ePrint Archive, Report 2011/615, December 2011. Online: https://eprint.iacr.org/2011/615 (fetched in May 2023)

[DIC18] F. Di Cerbo, A. Lunardelli, I. Matteucci, F. Martinelli, P. Mori, "A Declarative Data Protection Approach: From Human-Readable Policies to Automatic Enforcement", WEBIST (Revised Selected Papers) 2018: 78-98.

[DIE08] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246 (Proposed Standard), Internet Engineering Task Force, RFC Editor, August 2008.

[DIJ21] E. Dijk, C. Wang and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Internet-Draft draft-ietf-core-groupcomm-bis-08 (work in progress), IETF Secretariat, January 2023.

[DIN11] G. Dini and I. M. Savino, "LARK: A Lightweight Authenticated ReKeying Scheme for Clustered Wireless Sensor Networks", ACM Transaction on Embedded Computing Systems, vol. 10, no. 4, pp. 41:1–41:35, November 2011.

[DIN13] G. Dini and M. Tiloca, "HISS: A HIghly Scalable Scheme for Group Rekeying", The Computer Journal, Issue 56, Vol. 4, pp. 508–525, Oxford University Press, 2013.

[EDHOC-DEV] EDHOC Java Implementation. Online: https://github.com/rikard-sics/californium/tree/edhoc (fetched in May 2023)

[ERO05] P. Eronen and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279 (Proposed Standard), Internet Engineering Task Force, RFC Editor, December 2005.

[FAR13] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A.Keranen and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920 (Proposed Standard), RFC Editor, April 2013.

[FET11] I. Fette and A. Melnikov, "The WebSocket Protocol", RFC6455 (Proposed Standard), Internet Engineering Task Force, RFC Editor, December 2011

[FIE00] R. T. Fielding and R. N. Taylor, "Architectural styles and the design of network-based software architectures", Vol. 7, University of California, Irvine Doctoral dissertation, 2000.

[FIE22a] R. T. Fielding, M. Nottingham and J. Reschke, "HTTP Semantics", RFC 9110 (Internet Standard), Internet Engineering Task Force, RFC Editor, June 2022.

[FIE22b] R. T. Fielding, M. Nottingham and J. Reschke, "HTTP/1.1", RFC 9112 (Internet Standard), Internet Engineering Task Force, RFC Editor, June 2022.

[GER22] S. Gerdes, O. Bergmann, C. Bormann, G. Selander and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", RFC 9202 (Proposed Standard), Internet Engineering Task Force, RFC Editor, August 2022.

[GOSC-DEVa] Implementation of Group OSCORE for Eclipse/Californium. Online: https://github.com/rikard-sics/californium/tree/group_oscore (fetched in May 2023)

[GOSC-DEVb] Implementation of Group OSCORE for Contiki-NG. Online: https://github.com/Gunzter/contiki-ng/ (fetched in May 2023)

[GUE22] F. Günther and M. Ilunga, "Careful with MAc-then-SIGn: A Computational Analysis of the EDHOC Lightweight Authenticated Key Exchange Protocol", December 2022, https://eprint.iacr.org/2022/1705 (fetched in May 2023)

[GUN21] M. Gunnarsson, J. Brorsson, F. Palombini, L. Seitz and M. Tiloca, "Evaluating the performance of the OSCORE security protocol in constrained IoT environments", in "Internet of Things", vol. 13, Elsevier, 2021.

[GUN22] M. Gunnarsson, K. M. Malarski, R. Höglund and M. Tiloca, "Performance Evaluation of Group OSCORE for Secure Group Communication in the Internet of Things", ACM Transactions on Internet of Things, ACM, 2022 (To appear).

[HAR12] D. Hardt, "The OAuth 2.0 Authorization Framework", RFC 6749 (Proposed Standard), RFC Editor, October 2012.

[HAR15] K. Hartke, “Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641 (Proposed Standard), RFC Editor, September 2015.

[HÖG23] R. Höglund and M. Tiloca, "Key Update for OSCORE (KUDOS)", Internet-Draft draft-ietf-core-oscore-key-update-04 (work in progress), IETF Secretariat, March 2023.

[JAC22] C. Jacomme, E. Klein, S. Kremer and M. Racouchot, "A comprehensive, formal and automated analysis of the EDHOC protocol", October 2022, https://hal.inria.fr/hal-03810102 (fetched in May 2023)

[JON15] M. Jones, J. Bradley and N. Sakimura, "JSON Web Token (JWT)", RFC 7519 (Proposed Standard), Internet Engineering Task Force, RFC Editor, May 2015.

[JON16] M. Jones, J. Bradley and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800 (Proposed Standard), Internet Engineering Task Force, RFC Editor, April 2015.

[JON18] M. Jones, E. Wahlstroem, S. Erdtman and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392 (Proposed Standard), Internet Engineering Task Force, RFC Editor, May 2018.

[JON20] M. Jones, L. Seitz, G. Selander, S. Erdtman and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747 (Proposed Standard), Internet Engineering Task Force, RFC Editor, March 2020.

[KOS23] M. Koster, A. Keränen and J. Jimenez, "A publish-subscribe architecture for the Constrained Application Protocol (CoAP)", Internet-Draft draft-ietf-core-coap-pubsub-12 (work in progress), IETF Secretariat, March 2023.

[KRA03] H. Krawczyk, "SIGMA - The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols (Long version)", June 2003. Online: https://www.iacr.org/archive/crypto2003/27290399/27290399.pdf (fetched in May 2023)

[LAN16] A. Langley, M. Hamburg and S. Turner, "Elliptic Curves for Security", RFC 7748 (Informational), Internet Research Task Force, RFC Editor, January 2016.

[LAZ10] A. Lazouski, F. Martinelli, and P. Mori. "Usage control in computer security: A survey", Computer Science Review, 4(2):81–99, Elsevier, May 2010.

[LOD13] T. Lodderstedt, S.Dronia and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009 (Proposed Standard), Internet Engineering Task Force, RFC Editor, August 2013.

[MAT23] J. P. Mattsson, G. Selander, S. Raza, J. Höglund and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Internet-Draft draft-ietf-cose-cbor-encoded-cert-05 (work in progress), IETF Secretariat, January 2023.

[M-ECC] Micro-ECC: https://github.com/kmackay/micro-ecc (fetched in June 2023)

[MON-CY] Monocypher - Boring crypto that simply work: https://monocypher.org/ (fetched in June 2023)

[NOR20] K. Norrman, V. Sundararajan and A. Bruni, "Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices", September 2020, https://arxiv.org/abs/2007.11427 (fetched in May 2023)

[OMA-CORE] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Core, Approved Version 1.2, OMA-TS-LightweightM2M_Core-V1_2-20201110-A", November 2020. Online: http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf (fetched in May 2023)

[OMA-TP] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Transport Bindings, Approved Version 1.2, OMA-TS-LightweightM2M_Transport-V1_2-20201110-A", November 2020. Online: http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Transport-V1_2-20201110-A.pdf (fetched in May 2023)

[OSC-DEV] Implementation of OSCORE for Contiki-NG. Online: https://github.com/Gunzter/contiki-ng/ (fetched in May 2023)

[PAL22] F. Palombini, L. Seitz, G. Selander and M. Gunnarsson, "The Object Security for Constrained RESTful Environments (OSCORE) Profile of the Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9203 (Proposed Standard), Internet Engineering Task Force, RFC Editor, August 2022.

[PAL23a] F. Palombini, C. Sengul and M. Tiloca, "Publish-Subscribe Profile for Authentication and Authorization for Constrained Environments (ACE)", Internet-Draft draft-ietf-ace-pubsub-profile-06 (work in progress), IETF Secretariat, March 2023.

[PAL23b] F. Palombini, M. Tiloca, R. Höglund, S. Hristozov and G. Selander, "Using EDHOC with CoAP and OSCORE", Internet-Draft draft-ietf-core-oscore-edhoc-07 (work in progress), IETF Secretariat, March 2023.

[PAR04] J. Park and R. S. Sandhu, "The UCONABC usage control model", ACM Transactions on Information and System Security, 7(1): 128-174, ACM, February 2004.

[POS80] J. Postel, "User Datagram Protocol", RFC 768 (Internet Standard), Internet Engineering Task Force, RFC Editor, August 1980.

[POS81] J. Postel, "Transmission Control Protocol", RFC 793 (Internet Standard), Internet Engineering Task Force, RFC Editor, September 1981.

[RAH14] A. Rahman and E. Dijk, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390 (Experimental), Internet Engineering Task Force, RFC Editor, October 2014.

[RaspberryPi] Raspberry Pi Model 3 B. Online: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (fetched in May 2023)

[RES03] E. Rescorla and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", RFC 3552 (Best Current Practice), Internet Engineering Task Force, RFC Editor, July 2003.

[RES12] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347 (Proposed Standard), Internet Engineering Task Force, RFC Editor, January 2012.

[RES18] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, Internet Engineering Task Force, RFC Editor, August 2018.

[RES21] E. Rescorla, H. Tschofenig and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Internet Draft draft-ietf-tls-dtls13-43 (work in progress), IETF Secretariat, April 2021.

[SCH22a] J. Schaad, " CBOR Object Signing and Encryption (COSE): Structures and Process", RFC 9052 (Internet Standard), RFC Editor, October 2022.

[SCH22b] J. Schaad, "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053 (Informational), RFC Editor, October 2022.

[SCH22c] J. Schaad, "CBOR Object Signing and Encryption (COSE): Countersignatures", RFC 9338 (Internet Standard), RFC Editor, December 2022.

[SEI22] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200 (Proposed Standard), Internet Engineering Task Force, RFC Editor, August 2022.

[SEL19] G. Selander, J. P. Mattsson, F. Palombini and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC8613 (Proposed Standard), Internet Engineering Task Force, RFC Editor, July

2019.

[SEL23a] G. Selander, J. P. Mattsson and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Internet-Draft draft-ietf-lake-edhoc-19 (work in progress), IETF Secretariat, February 2023.

[SEL23b] G. Selander, J. P. Mattsson, M. Tiloca and R. Höglund, " Ephemeral Diffie-Hellman Over COSE (EDHOC) and Object Security for Constrained Environments (OSCORE) Profile for Authentication and Authorization for Constrained Environments (ACE)", Internet-Draft draft-ietf-ace-edhoc-oscore-profile-01 (work in progress), IETF Secretariat, March 2023.

[SHE12] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format", RFC 6690 (Proposed Standard), Internet Engineering Task Force, RFC Editor, August 2012.

[SHE14] Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252 (Proposed Standard), Internet Engineering Task Force, RFC Editor, June 2014.

[TDTLS] Eclipse TinyDTLS: https://github.com/eclipse/tinydtls (fetched in June 2023)

[THO21] E. Thormarker, "On using the same key pair for Ed25519 and an X25519 based KEM", April 2021. Online: https://eprint.iacr.org/2021/509 (fetched on May 2023)

[THO22] M. Thomson and C. Benfield, "HTTP/2", RFC 9113 (Proposed Standard), Internet Engineering Task Force, RFC Editor, June 2022.

[TI-SIM] Texas Instruments LAUNCHXL-CC1352R1 SimpleLink™ Multi-Band CC1352R Wireless MCU LaunchPad™ Development Kit: https://www.ti.com/tool/LAUNCHXL-CC1352R1 (fetched in June 2023)

[TIL16] M. Tiloca and G. Dini, "GREP: a Group REkeying Protocol based on member join history", in IEEE ISCC 2016, pp. 326-333, IEEE, 2016.

[TIL20] M. Tiloca, G. Dini, K. Rizki and S. Raza, "Group rekeying based on member join history", International Journal of Information Security, Volume 19, pp. 343-381, Springer, 2020

[TIL23a] M. Tiloca, J. Park and F. Palombini, "Key Management for OSCORE Groups in ACE", Internet-Draft draft-ietf-ace-key-groupcomm-oscore-16 (work in progress), IETF Secretariat, March 2023.

[TIL23b] M. Tiloca, R. Höglund, P. van der Stok and F. Palombini, "Admin Interface for the OSCORE Group Manager", Internet-Draft draft-ietf-ace-oscore-gm-admin-08 (work in progress), IETF Secretariat, March 2023.

[TIL23c] M. Tiloca, C. Amsüss and P. van der Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Internet-Draft draft-tiloca-core-oscore-discovery-13 (work in progress), IETF Secretariat, March 2023.

[TIL23d] M. Tiloca, R. Höglund, C. Amsüss and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Internet-Draft draft-ietf-core-observe-multicast-notifications-06 (work in progress), IETF Secretariat, April 2023.

[TIL23e] M. Tiloca and E. Dijk, "Proxy Operations for CoAP Group Communication", Internet-Draft draft-tiloca-core-groupcomm-proxy-08 (work in progress), IETF Secretariat, February 2023.

[TIL23f] M. Tiloca and R. Höglund, "OSCORE-capable Proxies", Internet-Draft draft-tiloca-core-oscore-capable-proxies-06 (work in progress), IETF Secretariat, April 2023.

[TIL23g] M. Tiloca, R. Höglund, L. Seitz and F. Palombini, "Group OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Internet Draft draft-tiloca-ace-group-oscore-profile-10 (work in progress), IETF Secretariat, March 2023.

[TIL23h] M. Tiloca, F. Palombini, S. Echeverria and G. Lewis, "Notification of Revoked Access Tokens in the Authentication and Authorization for Constrained Environments (ACE) Framework", Internet Draft draft-ietf-ace-revoked-token-notification-06 (work in progress), IETF Secretariat, June 2023.

[TIL23i] M. Tiloca, G. Selander, F. Palombini, J. P. Mattsson and J. Park, " Group Object Security for Constrained RESTful Environments (Group OSCORE)", Internet-Draft draft-ietf-core-oscore-groupcomm-18 (work in progress), IETF Secretariat, June 2023.

[VUC22] M. Vučinić, G. Selander, J. P. Mattsson, T. Watteyne "Lightweight Authenticated Key Exchange with EDHOC". Online: https://hal.inria.fr/hal-03434293/file/vucinic22lightweight.pdf (fetched in May 2023)

[WAL99] D. Wallner, E. Harder and R. Agee, "Key Management for Multicast: Issues and Architectures", Internet Engineering Task Force, RFC 2627 (Informational), RFC Editor, June 1999.

[WON00] C. K. Wong, M. Gouda and S. S. Lam, "Secure group communications using key graphs", IEEE/ACM Transactions on Networking, Issue 8, Vol. 1, pp. 16–30, IEEE/ACM, 2000.

[WOU14] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Internet Engineering Task Force, RFC 7250 (Proposed Standard), RFC Editor, January 2014.

[XAC13] OASIS. Oasis eXtensible Access Control Markup Language (XACML) version 3, OASIS standard. Technical report, 22 January 2013.

[ZHA05] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park, "Formal model and policy specification of usage control", ACM Transactions on Information and System Security, 8(4):351–387, ACM, November 2005.

[ZOL-FF] Zolertia Firefly: https://zolertia.io/product/firefly/ (fetched in June 2023)

Annex A: Glossary

Acronym	Definition
AAD	Additional Authenticated Data
ACE	Authentication and Authorization for Constrained Environments
AEAD	Authenticated Encryption with Associated Data
AM	Attribute Manager
AS	Authorization Server
BS	Bootstrap Server
CBOR	Concise Binary Object Representation
CH	Context Handler
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
COSE	CBOR Object Signing and Encryption
CPU	Central Processing Unit
DHT	Distributed Hash Table
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
EDHOC	Ephemeral Diffie-Hellman Over COSE
EAD	External Authorization Data
GM	Group Manager
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
JSON	Javascript Object Notation
KDC	Key Distribution Center
KUDOS	Key Update for OSCORE
LAKE	Lightweight Authenticated Key Establishment
LwM2M	Lightweight Machine-to-Machine
M2M	Machine-to-Machine (communications)
MAC	Message Authentication Code
MiTM	Man in The Middle
OMA	Open Mobile Alliance
OSCORE	Object Security for Constrained RESTful Environments
PSK	Pre-Shared Key
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RBAC	Rule Based Access Control
REST	Representational State Transfer
RD	Resource Directory
RPK	Raw Public Key

RS	Resource Server
SIFIS-Home	Secure Interoperable Full Stack Internet of Things for Smart Home
SM	Session Manager
SW	Software
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UCON	Usage Control
UCP	Usage Control Policy
UCS	Usage Control System
UDP	User Datagram Protocol
VM	Virtual Machine
WG	Working Group
WP	Work Package
XACML	eXtensible Access Control Markup Language

Annex B: Summary of Mapping against Architecture and Requirements

This annex provides an overview of how the security solutions presented in this document are mapped against the requirements defined in the deliverable D1.2 [D1.2] and the SIFIS-Home architecture components defined in the deliverable D1.4 [D1.4]. For each security solution, this information is also compiled at the beginning of the corresponding section, as an italic header.

The column of the table below specifies the following information.

Security solution: The security solution developed in WP3 and presented in this document.
Section: The section of this document where the security solution is presented.
F-REQ: List of Functional Requirements pertaining to the security solution.
NF-REQ: List of Non-Functional Requirements pertaining to the security solution.
SEC-REQ: List of Security Requirements pertaining to the security solution.
Components: List of the SIFIS-Home architecture components pertaining to the security solution.

Security solution	**Section**	**F-REQ**	**NF-REQ**	**SEC-REQ**	**Components**
Group OSCORE	4.1	N/A	PE-28, PE-29, P-30, P-31, P-32	SE-30, SE-32, SE-33, SE-34, SE-35	"Secure Message Exchange Manager" and "Content Distribution Manager" ("Secure Communication Layer" module)
OSCORE Profile of ACE	5.1	N/A	PE-28, PE-29,	SE-19, SE-30, SE-35, SE-40	"Authentication Manager" and "Key Manager" ("Secure Lifecycle Manager" module); "Secure Message Exchange Manager" and "Content

					Distribution Manager" ("Secure Communication Layer" module)
Notification of Revoked Access Credentials	5.2	N/A	PE-28, PE-29	SE-19, SE-40	"Authentication Manager" ("Secure Lifecycle Manager" module); "Secure Message Exchange Manager" and "Content Distribution Manager" ("Secure Communication Layer" module)
Usage Control Framework	5.3	F-30, F-31, F-32, F-33, F-34, F-35, F-47, F-48, F-49, F-50, F-51, F-52, F-53	P-19, PE-20, PE-21, PE-28, PE-29, US-15, US-16	SE-19, SE-40, SE-42	"Authentication Manager" ("Secure Lifecycle Manager" module)
Combined Enforcement of Access and Usage Control	5.4	N/A	PE-28, PE-29	SE-19, SE-40	"Authentication Manager" ("Secure Lifecycle Manager" module); "Secure Message Exchange Manager" and "Content Distribution Manager" ("Secure Communication Layer" module)
Key Provisioning for Group OSCORE using ACE	6.1	F-55, F-56, F-57	PE-28, PE-29, PE-31	SE-30, SE-31, SE-35, SE-36, SE-41, SE-44, SE-45	"Authentication Manager" and "Key Manager" ("Secure Lifecycle Manager" module); "Secure Message Exchange Manager" and "Content Distribution Manager" ("Secure Communication Layer" module)
EDHOC - Key Establishment for OSCORE	6.2	N/A	PE-28, PE-29, PE-36	SE-30, SE-31, SE-35, SE-43,	"Key Manager" ("Secure Lifecycle Manager" module); "Secure Message Exchange

				SE-47	Manager" and "Content Distribution Manager" ("Secure Communication Layer" module)

Annex C: Demo 1 – Secure Group Communication

This annex describes the focused demo "Secure Group Communication" from WP3, where devices exchange messages using CoAP for group communication, and protect communications in the group using the security protocol Group OSCORE (see Section 4.1).

A recording of the execution of this demo is available on the SIFIS-Home YouTube channel.

C.1 Considered Security Solutions

This demo covers multiple security solutions as part of a single storyline that unfolds throughout the demo execution. From a high-level point of view, such a storyline can be broken down into two different stages.

In the **first stage "Group Joining"**, devices securely join an OSCORE group by interacting with the responsible OSCORE Group Manager (GM). This stage relies on the method for securely joining OSCORE groups (see Section 6.1) based on the ACE framework for access control (see Section 3.8). In particular, the GM acts as an ACE Resource Server, while the joining devices act as ACE Clients. The OSCORE profile of ACE is used for secure communication between the joining devices and the OSCORE GM (see Section 5.1).

In the **second stage "Group Message Exchange"**, each device is member of an OSCORE group and securely communicates within that group. Members configured as CoAP client send requests over IP multicast within the respective group. The group members configured as a CoAP server reply with an individual response over unicast. All messages exchanged within the group are protected with Group OSCORE (see Section 4.1).

The storyline shown by the execution workflow of this demo has been built on the following two key points.

- Two OSCORE groups are considered, namely G1 and G2, to show different possible configurations. That is, Group G1 only uses the group mode of Group OSCORE to protect all messages exchanged in the group. Instead, Group G2 uses a combination of the two modes, i.e., the group mode is used to protect request messages, while the pairwise mode is used to protect response messages.

- The group members acting as CoAP servers are Raspberry Pi devices [RaspberryPi] that control physical actuators. In Group G1, the CoAP servers control lights, while in Group G2 they control mechanical locks. This ensures visual feedback, in addition to command line output and packet capture traces.

The software used for the demo is built on a foundation of available codebases that include Group OSCORE [GOSC-DEVa], the ACE framework [ACE-DEV], the OSCORE profile of ACE [ACE-DEV], and the joining process of OSCORE groups based on ACE [ACE-DEV].

C.2 Demo Description

Figure C.1 provides an overview of the setup considered in the demo.

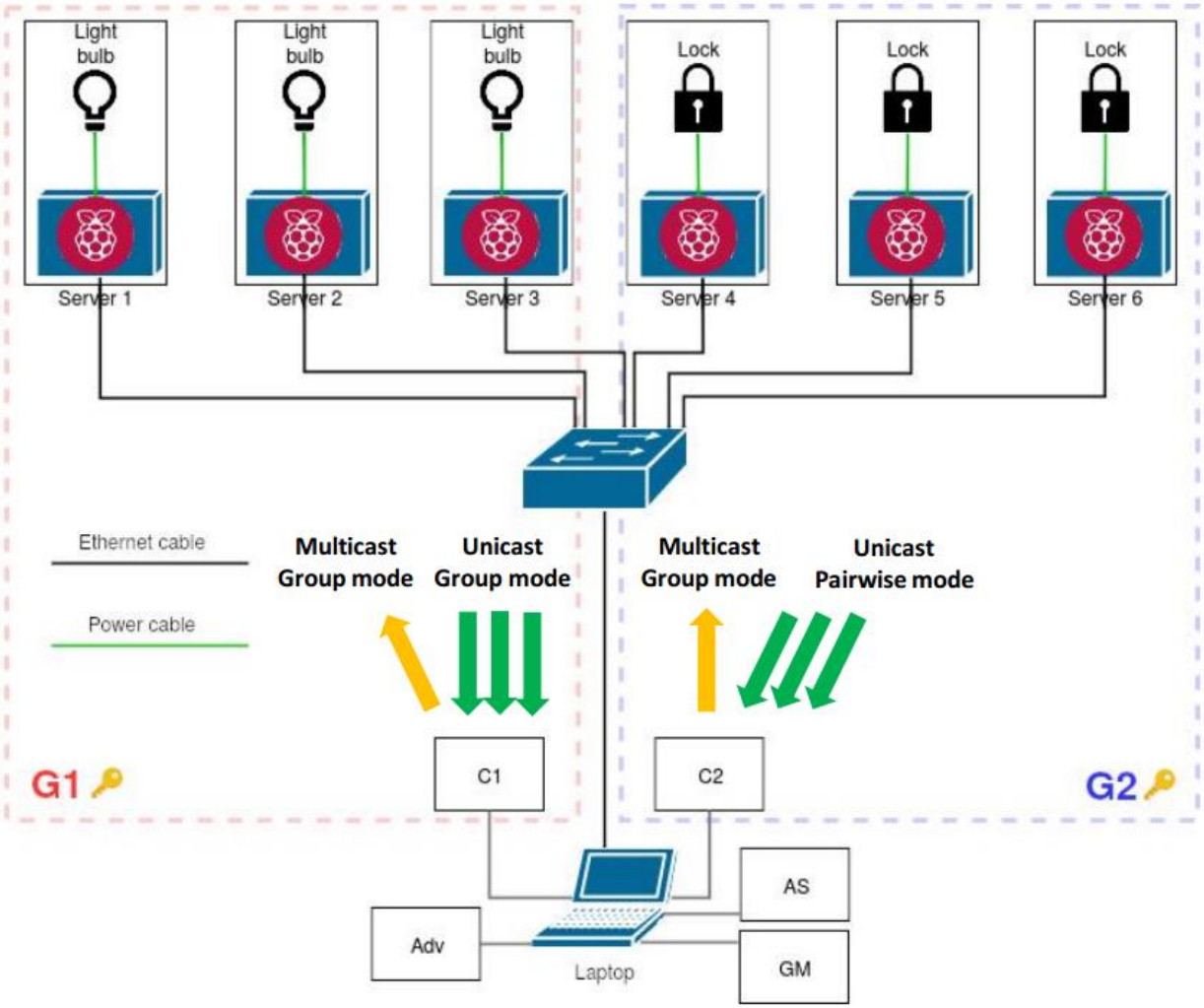


Figure C.1 - Overview of the demo "Secure Group Communication"

The Group G1 includes three CoAP servers, each of which is a Raspberry Pi device connected to a LED light. These servers are controlled by a CoAP client, namely C1, which sends its requests within the group over IP multicast and protects those with the group mode of Group OSCORE. The individual, unicast responses from the CoAP servers are also protected using the group mode of Group OSCORE.

The Group G2 includes three CoAP servers, each of which is a Raspberry Pi device connected to a mechanical lock. These servers are controlled by a CoAP client, namely C2, which sends its requests within the group over IP multicast and protects those with the group mode of Group OSCORE. In contrast to Group G1, the individual, unicast responses from the CoAP servers are protected using the pairwise mode of Group OSCORE, hence avoiding the use of digital signatures and yielding messages that are smaller in size.

Furthermore, a laptop is deployed and runs five processes, i.e.:

- The Client C1, which is a member of the Group G1 and controls the servers therein.

- The Client C2, which is a member of the Group G2 and controls the servers therein.
- The ACE Authorization Server (AS), which issues Access Tokens authorizing the different devices to join either Group G1 or Group G2.
- The OSCORE Group Manager (GM), as responsible for both Group G1 and Group G2.
- An external Adversary (Adv), which does not join either group and attempts to perform attacks against communications in the two groups, without success.

Initial settings

In order for the demo execution to be focused on the security solutions to demonstrate, the following pre-configurations have been considered and enforced.

All the prospective members of the two OSCORE groups are pre-registered at the AS, and are hence immediately able to request an Access Token to join an OSCORE group.

All the prospective group members, the AS, and the GM are configured to use the OSCORE profile of ACE when interacting throughout the ACE workflow for enforcing access control to join the OSCORE groups.

All the prospective group members are pre-configured with the following information:
- Their own public and private keys
- The resources at the group members acting as CoAP server and their respective URI
- The name of the OSCORE group that they are going to join and the associated IP multicast address
- The role in the OSCORE group that they are going to join
- The name of the Group Manager

Execution workflow

The execution workflow of the demo can be summarized in three key steps. The first two steps are part of the first stage “Group Joining”, while the third step is part of the second stage “Group Message Exchange”.

1. Each device obtains authorization evidence from the AS associated with the OSCORE GM, for joining Group G1 or Group G2 through the GM. The evidence is issued by the AS in the form of an Access Token.
2. Each device uploads the Access Token to the GM, establishes an OSCORE secure association with the GM per the OSCORE profile of ACE, and finally joins the correct OSCORE group through the GM.
3. Once both Group G1 and Group G2 are populated, the group members configured as CoAP client send actuation commands within the group they belong to, over IP multicast. The group members configured as CoAP server receive such requests, perform the requested action on the associated physical actuator, and then reply back over unicast. All the exchanged messages are protected with Group OSCORE.

The following provides a more detailed, step-by-step description of the demo execution workflow, with particular reference to Group G1.

Each of the servers Server 1 (S1), Server 2 (S2), and Server 3 (S3) in Figure C.1 performs the following steps.
1. It obtains an Access Token from the AS, as evidence of authorization to join Group G1.
2. It uploads the Access Token to the GM, and establishes an OSCORE secure association with the GM, as per the OSCORE profile of ACE.
3. It interacts with the GM by performing the joining procedure for joining Group G1. During this process, it provides the GM with its own public authentication credential. The GM provides the parameters and

keying material required to securely communicate within Group G1 using Group OSCORE.

The client C1 in Figure C.1 similarly performs the following steps.

1. It obtains an Access Token from the AS, as evidence of authorization to join Group G1.
2. It uploads the Access Token to the GM, and establishes an OSCORE secure association with the GM, as per the OSCORE profile of ACE.
3. It interacts with the GM by performing the joining procedure for joining Group G1. During this process, it provides the GM with its own public authentication credential. The GM provides the parameters and keying material required to securely communicate within Group G1 using Group OSCORE. In addition, it retrieves the public authentication credentials of the other group members.

The group members S1, S2, and S3 can retrieve the authentication credentials of C1 from the GM. Normally, this would be done through a dedicated interaction with the GM. However, for the sake of simplicity of the demo, the servers are all pre-configured with the public authentication credential of C1.

At this point, the stage “Group Joining” is completed and secure group communication within Group G1 can occur, i.e., the second stage “Group Message Exchange” begins.

That is, C1 can send group requests over IP multicast to the address associated with Group G1. Such requests are protected with the group mode of Group OSCORE and convey commands to change the status of the lights controlled by the servers S1, S2, and S3. In particular, C1 can issue the following commands, as text string specified as payload of the sent request: the "on" command switches the lights on, while the "off" command switches the lights off. The command to send is specified by the user through a command line interface.

Each server replies to C1, confirming the new, updated status of the light it controls as payload of the response message, which is sent over unicast to C1 and is protected with the group mode of Group OSCORE. The content of the response received by C1 is displayed to the user on the display of the laptop where C1 runs.

The same execution flow occurs in Group G2, with the following differences: the specifically considered devices are C2, Server 4 (S4), Server 5 (S5), and Server 6 (S6) in Figure C.1; the group requests that C2 sends over IP multicast to the address associated with Group G2 convey commands to close or open the locks controlled by the servers S4, S5, and S6, through a text string “lock” or “unlock” specified as payload of the sent request; the response messages from S4, S5, and S6 are protected with the pairwise mode of Group OSCORE.

Finally, the demo also shows how the external adversary (Adv) attempts to perform two different attacks against the groups, with no success. The following considers the Group G1, but the same principle holds if the attack is mounted against the Group G2.

The first attack is a replay attack, where Adv replays an old, valid group message, by sending it to the servers in the group. The demo shows that the attack fails, since the servers S1, S2, and S3 are indeed able to detect the message to be a replay as expected when using Group OSCORE, and thus discard it.

In the second attack, Adv crafts a new, semantically valid message and injects it into Group G1. The demo shows that the attack fails, since Adv does not have the required keying material to correctly protect the injected message, which servers S4, S5, and S6 fail to successfully decrypt and hence discard.

Annex D: Demo 2 – EDHOC: Key Establishment for OSCORE

This annex describes the focused demo "EDHOC: Key Establishment for OSCORE" from WP3, where two devices communicating with CoAP (see Section 3.1) first use the authenticated key establishment protocol EDHOC (see Section 6.2) for securely establishing an OSCORE Security Context, and then use OSCORE (see Section 3.7) to protect their communications end-to-end. The demo also shows the alternative, optimized EDHOC execution workflow presented in Section 6.2.1.

A recording of the execution of this demo is available on the SIFIS-Home YouTube channel.

D.1 Considered Security Solutions

This demo considers two devices communicating with one another, i.e., a CoAP client and a CoAP server. From a high level point-of-view, the demo execution consists of two (possibly overlapping) stages:

1. The execution of the EDHOC protocol between the CoAP client acting as EDHOC Initiator and the CoAP server acting as EDHOC Responder.
2. The exchange of CoAP messages between the CoAP client and the CoAP server, which protect their communications using the OSCORE Security Context derived from the EDHOC execution.

In order to demonstrate different configuration alternatives among those provided by EDHOC, four different settings have been considered, as specifically covering the following aspects.

In terms of peer authentication, the demo considers both peers using either: (A) signatures computed with private signing keys; or (B) Message Authentication Codes (MACs) computed with symmetric keys derived from static Diffie-Hellman keys.

In terms of EDHOC execution, the demo considers either: (C) the execution of EDHOC and the establishment of the OSCORE Security Context, strictly followed by the exchange of OSCORE-protected data messages (see Figure 6.2 in Section 6.2.1); or (D) the shortened, optimized execution of EDHOC, where a single optimized request combines the last EDHOC message with the first OSCORE-protected data message (see Figure 6.3 in Section 6.2.1).

Consistently, the following four combinations have been considered as different demo configurations.

1. (A+C): peer authentication with signatures and private signing keys, with no optimized execution
2. (A+D): peer authentication with signatures and private signing keys, with the optimized execution
3. (B+C): peer authentication with MACs and static Diffie-Hellman keys, with no optimized execution
4. (B+D): peer authentication with MACs and static Diffie-Hellman keys, with the optimized execution

The software used for the demo is built on a codebase available at [EDHOC-DEV], which also includes support for the optimized EDHOC execution workflow based on the optimized request.

D.2 Demo Description

Figure D.1 provides an overview of the setup and message flow considered in the demo.

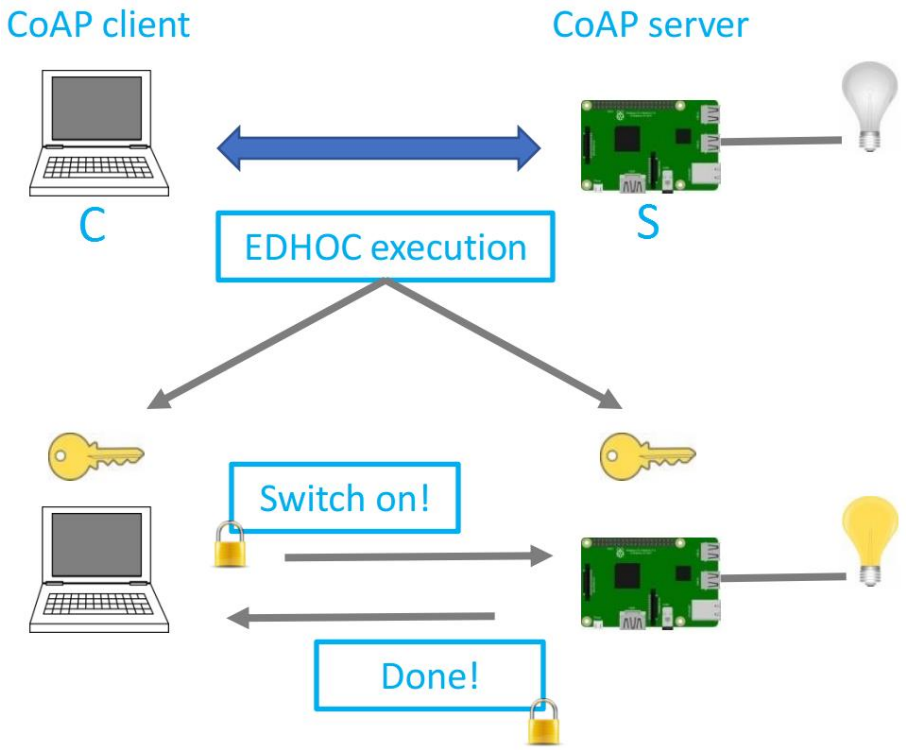


Figure D.1 - Overview of the demo "EDHOC: Key Establishment for OSCORE"

The demo setup includes two hardware components: a laptop (C) and a Raspberry Pi (S) [RaspberryPi]. In particular, C acts as CoAP client and EDHOC Initiator, while S acts as CoAP server and EDHOC Responder. Furthermore, S is connected to a LED light, acting as an actuator that physically controls the state of the light.

This setup relies on a one-to-one communication model, where C can control S by sending CoAP requests to S over IP unicast and protected with OSCORE. After S has successfully decrypted and processed an incoming request from C - and potentially altered the state of the LED light as indicated in the request - S sends a CoAP response to C, also protected with OSCORE and specifying the result of the performed operation. The protection of such exchanged data messages with OSCORE relies solely on the OSCORE Security Context that C and S have established as a result of the EDHOC execution.

The setup can be easily reconfigured for adhering to any of the configurations introduced in Section D.1.

Initial settings

The following pre-configurations have been considered and enforced on C and S, irrespective of the specifically used configuration from Section D.1.

Each peer is configured with its own pair of private and public key as well as with the public key of the other peer, to be used as authentication credentials during the execution of EDHOC. Furthermore, both peers are configured with the EDHOC cipher suite to use.

Additionally, C is pre-configured with the URI of the light resource hosted by S, where C can send its OSCORE-protected CoAP requests in order to switch on or off the light controlled by S. Finally, C is also pre-configured with the URI of the EDHOC resource hosted by S, thus enabling C to effectively act as EDHOC Initiator by sending the first EDHOC message to S in order to start an EDHOC execution.

Note that, on the other hand, no OSCORE Security Context is pre-configured on the two peers. Rather, those are going to establish a new one by executing the EDHOC protocol.

Execution workflow

With reference to the four configurations introduced in Section D.1, the following applies.

For all configurations: C acts as EDHOC Initiator; S acts as EDHOC Responder; the OSCORE-protected data requests from C include a payload according to which S sets the specified state of the LED light; the OSCORE-protected data response from S includes a payload that specify the new state of the LED light. In particular, the payload of the data requests from C is a text string, i.e., either "on" for switching the light on, or "off" for switching the light off. The user can provide such a command to C through a command line interface.

For configuration (1), EDHOC is first executed and run to completion, with peer authentication achieved by means of signatures through the two peers' private signing keys. After that, the two peers derive an OSCORE Security Context, from the security material established through the EDHOC execution. Finally, C sends to S an OSCORE-protected data request, in order to switch on the LED light. After S has processed the request, it replies to C and confirms that the LED light is on.

For configuration (2), EDHOC is executed like in configuration (1), with the difference that the optimized workflow is used. That is, C sends a single CoAP request combining together the last EDHOC message_3 and the OSCORE- protected data intended for S to switch on the light. When receiving the combined request, S first protects its EDHOC-related part and derives the OSCORE Security Context. Then, S uses that OSCORE Security Context to decrypt the OSCORE-protected part. After that, S proceeds with the processing of the request and the following reply to C, just like in configuration (1). This effectively saves 1 round trip compared to configuration (1).

For configuration (3), the same as in configuration (1) applies, with the difference that peer authentication is achieved by means of MACs computed through keying material derived from static Diffie-Hellman keys. This results in messages that are smaller in size, due to the difference in size between signatures and MACs.

For configuration (4), the same as in configuration (2) applies, with the difference that peer authentication is achieved by means of MACs computed through keying material derived from static Diffie-Hellman keys. From a different point-of-view, the same as in configuration (3) applies, with the difference that C and S rely on the optimized EDHOC execution. This effectively saves 1 round trip compared to configuration (3).

The demo effectively highlights the difference between the four different configurations above. Specifically, it makes it possible to easily notice and appreciate the differences in terms of: the size of the exchanged EDHOC messages; and the time required to complete an EDHOC execution and the first exchange of data protected with OSCORE. These depend on the used EDHOC authentication method (i.e., as based on signatures or on MACs) and on whether the optimized EDHOC execution workflow is used or not.

Furthermore, following the first OSCORE-protected data exchange, and irrespective of the considered configuration, the user can provide the "on" or "off" command to C through the command line interface. This results in C sending a new OSCORE-protected data request to S, which will process it and accordingly reply to C with an OSCORE-protected response.

The demo also considers a CoAP client that does not use EDHOC or OSCORE, and that sends an unprotected CoAP request to S, as an attempt to switch the state of the LED light. Since S admits the processing of requests

targeting the light resource only if protected with OSCORE, the requests from the rogue CoAP client are shown to be rejected and are not processed, with S simply replying to those with an unprotected error response.