D3.2

Initial Architecture Requirements Report

WP3 – Network and System Security

SIFIS-Home

Secure Interoperable Full-Stack Internet of Things for Smart Home

Due date of deliverable: 31/03/2022
Actual submission date: 31/03/2022

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

28/03/2022
Version 0.09

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: Andrea Saracino (CNR), Domenico De Guglielmo (MIND)

Revision History

Version	Date	Name	Partner	Section Affected Comments
0.1	07/10/2021	Document created, with TOC	RISE	All
0.2	05/12/2021	Mapping to D1.2 requirements. Mapping to D1.3 components.	RISE	All
0.3	06/12/2021	Executive summary; conclusions	RISE	All
0.4	08/12/2021	Introduction; updated references and glossary; minor fixes	RISE	All
0.5	14/12/2021	Overview of security solutions (except for CNR items)	RISE	All
0.6	25/12/2021	Background section (except for CNR items)	RISE	All
0.7.1	28/12/2021	Section on Group OSCORE	RISE	All
0.7.2	11/01/2022	Subsection on profiling EDHOC for CoAP and OSCORE	RISE	All
0.7.3	12/01/2022	Subsection on key limits and key update for OSCORE	RISE	All
0.7.4	24/01/2022	Subsections on proxies for group communication; CoAP responses over IP multicast; caching of OSCORE-protected responses	RISE	All
0.7.5	25/01/2022	Section on EDHOC	RISE	All
0.7.6	26/01/2022	Sections on OSCORE-capable proxies; notification of revoked access credentials; background content on dynamic evaluation of access policies	RISE, CNR	All
0.7.7	27/01/2022	Section on OSCORE and Group OSCORE profiles of ACE	RISE	All
0.7.8	28/01/2022	Sections on key provisioning for Group	RISE	All

		OSCORE using ACE; discovery of OSCORE groups; configuration of OSCORE groups; OSCORE and EDHOC in LwM2M		
0.7.9	09/02/2022	Sections on usage control framework; combined enforcement of access and usage control	CNR	All
0.7.9.1	11/02/2021	Typesetting	RISE, CNR	All
0.8	13/02/2021	Section on DoS counteraction	RISE	All
0.9	11/03/2022	Processed comments from internal review	RISE	All
1.0	30/03/2022	Finalized for submission	RISE	All

Executive Summary

This document is an outcome of WP3 "Network and System Security" and provides a preliminary description of the network and system security solutions designed and developed in the SIFIS-Home project. These are intended to be applicable especially – but not only – to the use cases and IoT-based Smart Home environment considered in the 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.

The security solutions have been designed and developed consistently with the requirements presented in the deliverable D1.2 "Final Architecture Requirements Report". In particular, 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.

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

Table of contents

1 Introduction

This document provides a preliminary description of the security solutions, mechanisms and approaches developed in Work Package 3 (WP3) "Network and System Security" during the first half of the SIFIS-Home project, i.e., up until March 2022.

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 and presents the security solutions pertaining to that 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 D1.3 "Initial Component, Architecture, and Intercommunication Design" [D1.3].

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.

Section 4 presents the security solutions under the activity area "Secure and Robust (Group) Communication for the IoT", whose work has been carried out under Task T3.1 "Secure, interoperable and robust communication". These security solutions comprise protocols and methods for end-to-end protected and robust communication with IoT devices, supporting also a group communication model and the presence of (untrusted) transport intermediaries such as proxies. The above especially includes the secure communication protocol Group OSCORE, as well as the use of the OSCORE and Group OSCORE protocols in environments using group communication and/or transport intermediaries.

Section 5 presents the security solutions under the activity area "Access and Usage Control for Server Resources", whose work has been jointly carried out under both Tasks T3.2 "Security Lifecycle Management" and T3.3 "Dynamic Multi-Domain Security and Safety Policy Handling". These security solutions comprise methods to enforce fine-grained access control of resources at their hosting server contextually with end-to-end secure communication, as well as techniques for dynamically evaluating access policies and consequently adjust/revoke stale access credentials. The above especially includes profiles of the ACE framework for end-to-end secure communication, as well as methods to achieve dynamic access and usage control of server resources, possibly within the ACE framework.

Section 6 presents the security solutions under the activity area "Establishment and Management of Keying Material", whose work has been carried out under Task T3.2 "Security Lifecycle Management". These security solutions comprise protocols and methods to distribute, establish and renew keying material, as especially intended for end-to-end protected message exchange also in group communication environments. The above

especially includes: the use of ACE to distribute keying material for Group OSCORE; the EDHOC protocol to establish OSCORE keying material among two peers; the KUDOS protocol to update current OSCORE keying material.

A final description of the security solutions designed and developed in WP3 will be provided in deliverable D3.3 "Final report on Network and System Security Solutions". This will be released in June 2023, and it will update and obsolete the present document, thus acting as final comprehensive description of WP3 activities.

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 below.

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 D1.3 "Initial Component, Architecture, and Intercommunication Design" [D1.3]. In particular, the security solutions developed in WP3 pertain to the SIFIS-Home architecture components within the “Secure Communication Layer” module and the “Secure Lifecycle Manager” module.

2.1 *Secure and Robust (Group) Communication for the IoT*

The work on this activity area occurs 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) [TIL21a] is currently under development 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 [SCH17], 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.

Proxies for CoAP group communication – The CoAP protocol natively supports the use of transport intermediaries, such as proxies, deployed between a client endpoint and a server endpoint. A proxy can, among other things, serve cached responses or perform protocol translation across different communication legs. When one-to-many group communication for CoAP is used [RAH14]][DIJ21], several processing steps and issues to address are left open at intermediaries. Activities on this topic have been defining how forward-proxies and reverse-proxies forward a group request to multiple servers, and then forward back the multiple individual responses to the origin client. Support must be ensured also in case group communications are protected end-to-end with Group OSCORE.

CoAP responses over IP multicast – The CoAP protocol provides the "Observe" feature [HAR15]. This allows a client endpoint to register its interest at a server endpoint’s resource, and to automatically receive notification responses upon changes in the resource representation. This has been recently enabled also in group communication scenarios [RAH14][DIJ21], where 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. Activities on this topic have been defining how a server can provide such functionality, by sending one single notification response targeting all the observer clients at once (e.g., over IP multicast). Clearly, support must be ensured also in case intermediary proxies are used, and in case group communications are protected end-to-end with Group OSCORE.

Caching of OSCORE-protected responses – As originally specified, the security protocol OSCORE does not make it possible to cache protected responses at intermediary proxies. That is, two identical plain requests result in two different OSCORE-protected requests, which thus never produce a cache hit. Activities on this topic have been enabling cacheability of OSCORE responses, building on the new concept of "deterministic request". In applications providing 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 improvements in terms of performance.

OSCORE-capable proxies – As originally specified, the security protocol OSCORE is intended to be used only between two "application endpoints", acting as origin CoAP client and origin CoAP server. At the same time, it is not intended to be used by possible intermediaries, such as transport proxies, deployed between the two application endpoints. That is, only the application endpoints are supposed to be also "OSCORE endpoints", and therefore to apply and consume the OSCORE protection to their exchanged messages. As motivated by a number of use cases, activities have been ongoing to define how OSCORE can also be used at OSCORE-capable proxies, i.e., at OSCORE endpoints that are not necessarily application endpoints. This in turn opens for multiple, nested protections of a same CoAP message, by applying multiple OSCORE protection layers in sequence. For instance, an origin CoAP client may want to achieve both end-to-end OSCORE protection with the origin server, as well as with the adjacent transport intermediary acting as next hop towards the origin server.

Robustness and resilience against Denial of Service – Denial of Service (DoS) attacks aim at making the targeted device unavailable for other devices trying to reach it, hence hindering the system from serving legitimate requests. This can be very effective in Internet of Things scenarios, where server devices might be constrained in terms of hardware resources and energy budget. Activities on this topic have been developing a solution to mitigate the impact of these attacks. This relies on a reactive, adaptive and host-based approach that takes as input information about ongoing attacks from already used secure communication layers and protocols, such as DTLS [RES21]. By assessing the severity of an ongoing attack, the victim device can react by trading service availability and quality of service against attack exposure. Under severe attack conditions, this can further leverage a trusted intermediary for holding and later relaying messages, as well as the use of low-power modes of operation to limit the attack impact on energy consumption.

2.2 *Access and Usage Control for Server Resources*

The work on this activity area occurs 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 and Group OSCORE profiles of ACE – The ACE framework for authentication and authorization in constrained environments (ACE) [SEI21] delegates to separate specifications the details about secure communication between the ACE entities, especially Clients and Resource Servers. Activities on this topic have been defining different profiles of ACE, to enable secure communication between Client and Resource Servers as based on i) OSCORE; or ii) Group OSCORE, when the Client is member of an OSCORE group and access control is enforced for accessing resources at the Resource Servers in the same OSCORE group. Both profiles provide mutual authentication of Client and Resource Server, as well as proof-of-possession of involved secret keys.

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 been designing 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.

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. Furthermore, 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.

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 for group OSCORE (distribution of group keying material; group configuration; discovery) – 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 responsible for driving the joining process of new authorized group members and for providing those with the required group keying material, as well as for providing possible assistance to current group members. Besides, two additional services are required to cover the full lifecycle of an OSCORE group. First, authorized Administrators must be able to create and configure OSCORE groups at the Group Manager. Second, just deployed devices must be able to discover an OSCORE group, and especially which Group Manager they should contact in order to join it as new group members.

EDHOC - Key establishment for OSCORE, including profiling and performance optimizations – The key establishment protocol Ephemeral Diffie-Hellman Over COSE (EDHOC) currently under development enables 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 to establish a Security Context that the two devices can use to protect their communications with OSCORE. Specific performance optimizations are also under development, especially the merging of, on 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.

Key limits and lightweight key update for OSCORE – The security protocol OSCORE provides application-

layer end-to-end protection between endpoints communicating with the CoAP protocol. Specifically, OSCORE uses AEAD algorithms to ensure integrity and confidentiality of the exchanged messages. A security analysis of such algorithms [GÜN21] has identified issues that, in the long run, can allow forgery attacks. Thus, limits must be considered as to how many times a certain key is used for encryption, or how many failed decryptions should be tolerated for one key. When these limits are exceeded, further use of the same key can break the security properties of the used AEAD algorithm. Activities on this topic have focused on i) defining appropriate key usage limits when using OSCORE and how to take them into account; and ii) defining an efficient and lightweight procedure for two OSCORE endpoints to update OSCORE keying material, while preserving Forward Secrecy.

Use of OSCORE and EDHOC in the LwM2M management framework – The standard OMA Lightweight Machine-to-Machine (LwM2M) [OMA-CORE] defines a framework for configuring, monitoring and controlling IoT devices, namely LwM2M Clients. This relies on a Bootstrap Server (BS) assisting the LwM2M Client in its early enrollment phase, and on a LwM2M Server acting as Device Manager where the LwM2M Client "registers" after bootstrapping. The LwM2M standard relies on the CoAP protocol as the main option for transporting messages and supports OSCORE as the end-to-end secure communication protocol at the application level [OMA-TP]. Typically, the BS provides the LwM2M Client with information to derive an OSCORE Security Context to use with the LwM2M Server. This has limited flexibility and does not ensure Forward Secrecy of the OSCORE keying material. Activities on this topic have focused on defining how, by following indications from the BS, a LwM2M Client can establish an OSCORE Security Context with the LwM2M server by executing the key establishment protocol EDHOC, thus overcoming the limitations above.

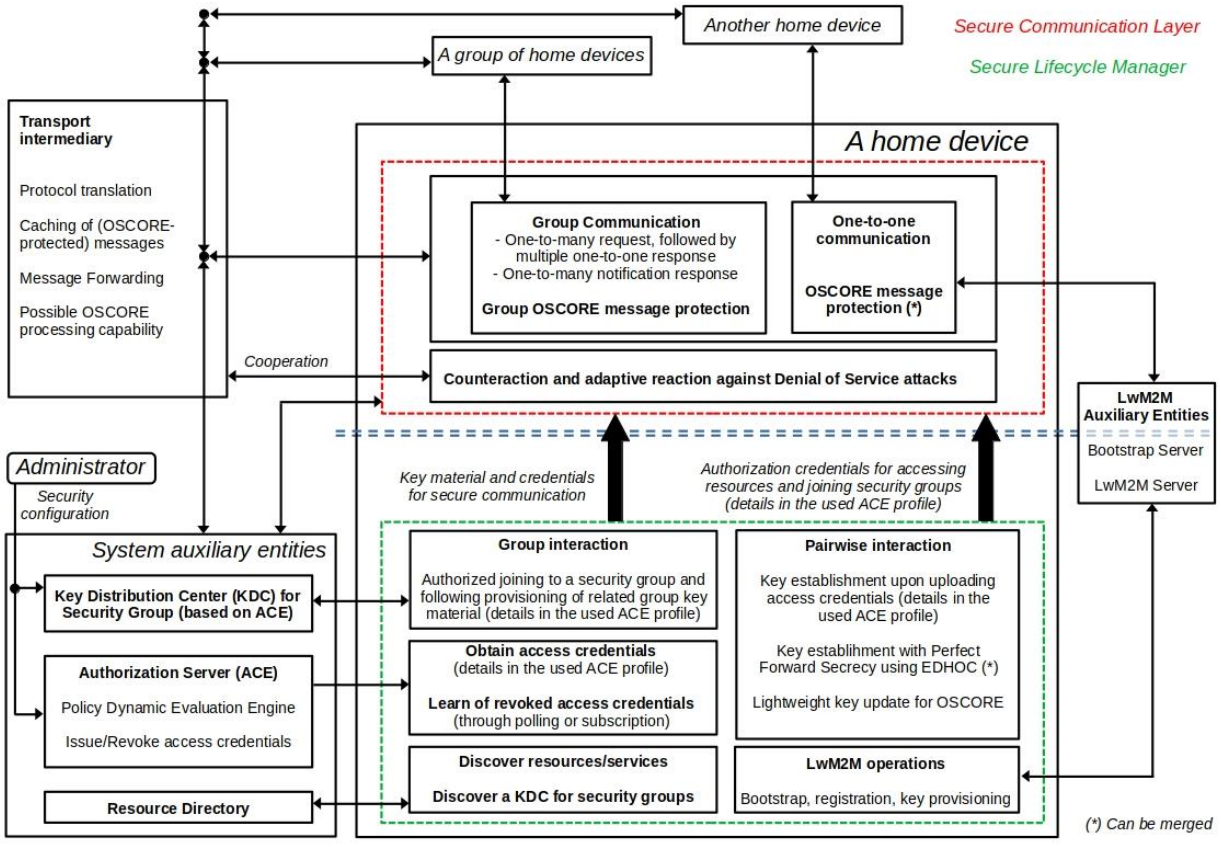

Figure 2.1 - Overview of the network & system security solutions and their relation

Figure 2.1 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 in Figure 2.1.

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. An external intermediary can be deployed between sender and recipient endpoints, as a transport proxy and facilitator in counteracting DoS attacks. In such a case, this proxy can preserve cacheability of OSCORE-protected response message and may be able to actively use OSCORE.

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; 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; iii) methods for discovery, among other endpoints and resources, a Key Distribution Center (KDC) acting as Group Manager for a security group where Group OSCORE is used.

A number of security solutions related to security lifecycle management is facilitated by auxiliary entities, typically in control of a privileged Administrator. These are, for instance: i) the KDC responsible for one or multiple security groups; ii) an AS responsible to issue access credentials according to pre-installed access policies; iii) a Resource Directory that facilitates the discovery of links to server devices and their resources; and iv) a pair of Bootstrap Server and LwM2M Server, with which a device acting as LwM2M Client interacts throughout the Bootstrapping and Registration process as well as follow-up exchanges.

3 Background Concepts and Technologies

This section introduces background concepts and technologies referred hereafter.

3.1 *CoAP*

The Constrained Application Protocol (CoAP) [SHE14] is an application layer, web transfer protocol based on 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 (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.

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).

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) [TIL21a] must be used as security protocol (see Section 4.1).

3.3 *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.

3.4 *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

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.

3.5 *CBOR and COSE*

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) [SCH17] 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.

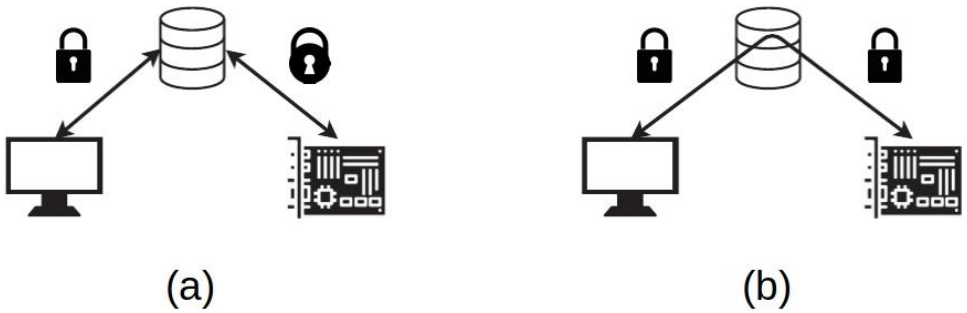

Figure 3.1 - Hop-by-hop vs. end-to-end security

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 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.

Figure 3.1(b) shows the 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 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.

3.7 *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 [SCH17] 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 [PAL21a] (see Section 5.1) and [SEL21] (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 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.

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.

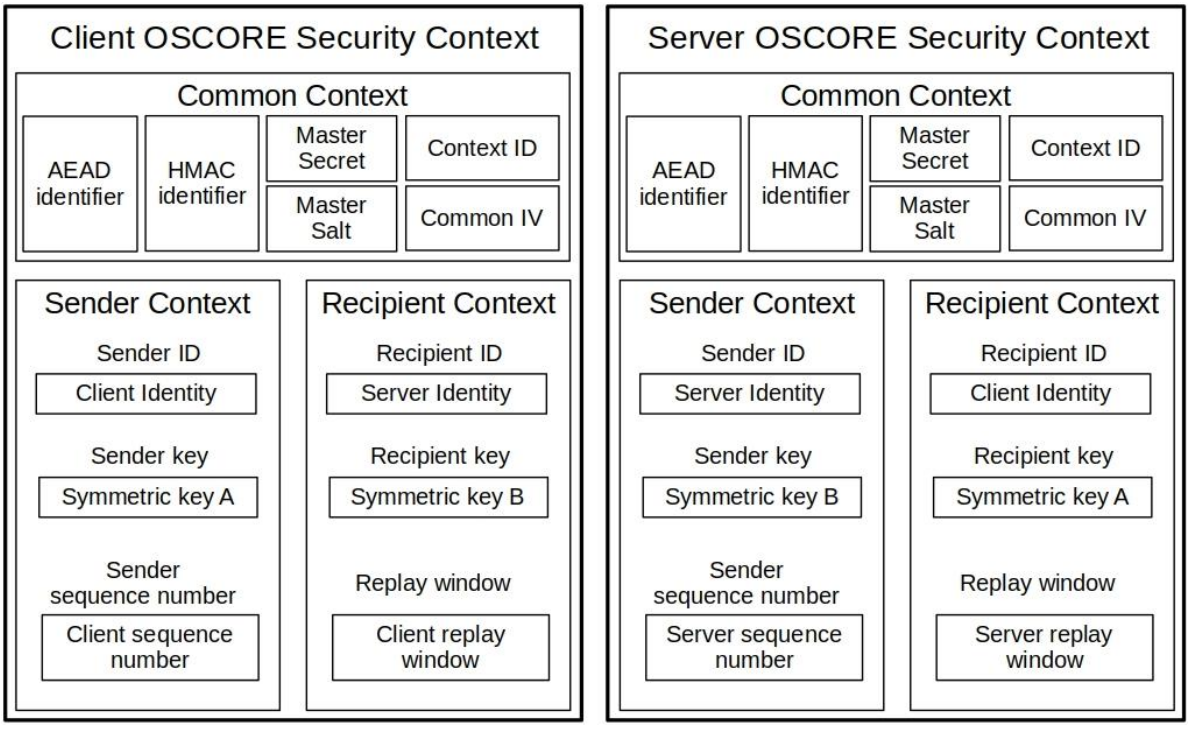

Figure 3.2 - OSCORE Security Contexts for a Client and Server pair (only the fields used during operation)

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;
- the Master Salt, an optional byte string used with the Master Secret to derive the keys and IVs;
- a Context ID, 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 Context ID, 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 or 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.

Version	Type	Token Length	CoAP-Code	Message ID
Token				
Option A		Option B	Option C	Option D
Payload delimiter	CoAP-Payload			

(a) CoAP message format.

Version	Type	Token Length	CoAP-Code	Message ID
Token				
Option B	OSCORE Option	Payload delimiter		
Encrypted{Option A, Option C, Option D, CoAP-Code, CoAP-Payload} + AEAD-tag				

(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 [SCH17]. 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 used as payload of the protected CoAP message, and any encrypted options are removed from the CoAP message. 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 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.

3.8 *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 [SEI21] 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 [SCH17], 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 [BOR13], 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.

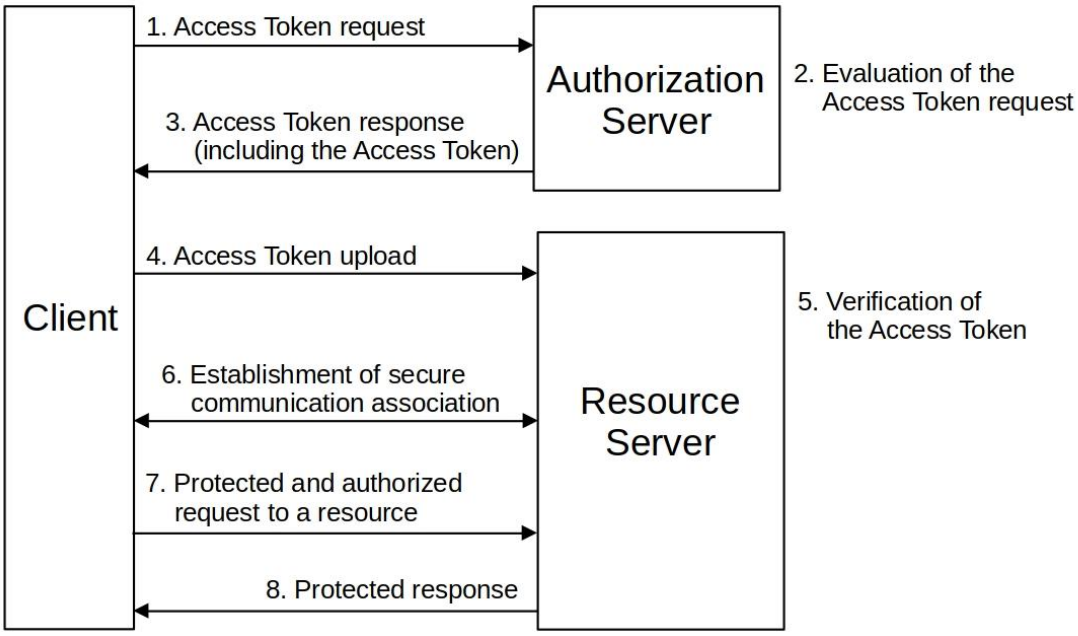

Figure 3.4 - ACE Framework workflow.

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 [GER21] 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 communicates 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.

3.9 *Dynamic evaluation of access policies*

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.

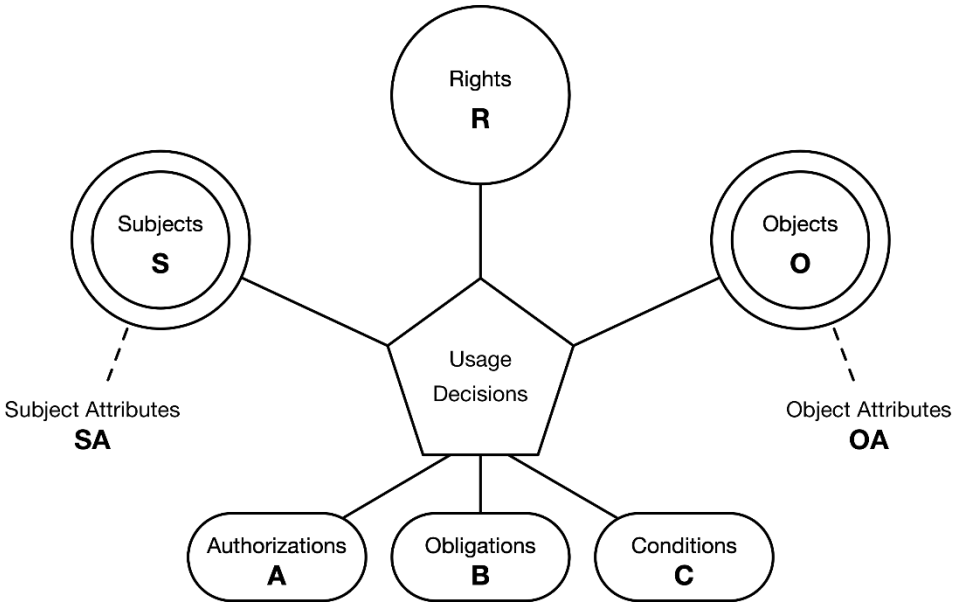

Figure 3.5 - Components of the Usage Control Model

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 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 [D1.3]:

- *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 security protocol to use. However, in group communication scenarios this results in the following issues.

First, just 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) [TIL21a] 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. 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 pairwise keys derived from asymmetric, individual keying material (see Section 4.1.5).

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.

Group OSCORE defines two different modes, as different ways to protect CoAP messages. It is up to the application to 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.

Finally, just as OSCORE, Group OSCORE provides message binding of responses to requests, which in turn provides relative freshness of responses, and 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 (s)he is also a member of both such groups.

A Java implementation of Group OSCORE from RISE is currently under development 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.

A Contiki-NG implementation of Group OSCORE from RISE is currently under development at [GOSC-DEVb], as intended to be integrated in the Contiki-NG operating system [Contiki-NG]. An 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, 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. Finally, Section 4.1.5 describes the pairwise mode of Group OSCORE.

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 keys of endpoints joining a group, 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 key 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 [SEI21] (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 supports 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 optional and relevant only if the group mode is used. The elements marked with (^) are optional and relevant only if the pairwise mode is used.

- One Common Context, shared by all the endpoints in the OSCORE group. The following new parameters are included in the Common Context: the public key of the Group Manager; the Signature Encryption Algorithm; the Signature Algorithm; and the Group 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.
- 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 key of the associated endpoint as well as with the Pairwise Recipient Key to use with that endpoint.

Context component	**New information element**
Common Context	Group Manager Public Key * Signature Encryption Algorithm * Signature Algorithm * Group Encryption Key ^ Pairwise Key Agreement Algorithm
Sender Context	Endpoint's own private key ^ Pairwise Sender Keys for the other Endpoint
Each Recipient Context	Public key 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 ID Context parameter in the Common Context contains the Group Identifier (Gid) of the OSCORE group, which is thus used as Context ID 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 public key 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).

Signature Encryption Algorithm 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 section 16.4 of [SCH17]).

A corresponding Pairwise Key Agreement Algorithm is used to derive pairwise symmetric keys, when using the pairwise mode of Group OSCORE (see Section 4.1.5). The encryption algorithm used to protect messages with the pairwise mode is indicated by the AEAD Algorithm field inherited from the OSCORE Security Context original format.

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

The Group 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 and Common IV – 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 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 key of the associated endpoint. The public key is used to verify the signature of the received OSCORE message when using the group mode (see Section 4.1.4), or to derive a pairwise key for verifying OSCORE messages from the associated endpoint protected with the pairwise mode (see Section 4.1.5).

The input parameters for deriving the Recipient Context parameters and the public key of the associated endpoint 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 endpoint in the OSCORE group. The received message, together with the Common Context, includes everything necessary to derive a security context for verifying a message, except for the public key of the associated 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 key. Such devices may instead be configured to drop a received message for which there is currently no Recipient Context, and retrieve the public key 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 Appendix A.2 of [SCH17], 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 key of the sender endpoint and the public key 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 process.

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 Group 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 [TIL21b].

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 Group Encryption Key.
- If CoAP Observe [HAR15] is supported, for each newly started observation, the client has to store the values of the Gid and of its own Sender ID at the moment, used as 'kid context' and 'kid' parameter in the original Observe 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 observation 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 key. 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 Group 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.
- If CoAP Observe [HAR15] is supported, for each newly started observation, the server stores the values of the 'kid context' and 'kid' parameters from the original Observe request, i.e., the Gid and the Sender ID of the observer client at that time. Then, the server does not update those stored values, even if the observer 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.
- 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 Group Encryption Key.
- If CoAP Observe [RFC7641] is supported, the server may have ongoing observations, started by Observe 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 notifications with its own Sender Context from that new Security Context.
    - For each ongoing observation, the server should include in the first notification protected with the new Security Context also the 'kid context' parameter, which has a 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 notifications for those observations.
    - For each ongoing observation, the server has to use the stored values of the 'kid context' and 'kid' parameters from the original Observe request, i.e., the Gid and the Sender ID of the observer client at the time of the original Observe request, as value for the 'request_kid_context' and 'request_kid' elements in the AAD (see Section 4.1.2.3), when protecting notifications for that observation.

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 group request. In such a case, 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 as AEAD nonce for the encryption process.
- Must increment the Sender Sequence Number by one;
- Must include in the response the 'Partial IV' parameter, which is set to the Partial IV above.
- 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 has to check that the replying server is the expected one, by relying on the server's public key 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 key. 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 Group 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 server may reply with a 4.00 (Bad Request) 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.

If CoAP Observe [HAR15] is supported, for each ongoing observation, the client has to use the stored values of the 'kid context' and 'kid' parameters from the original Observe request, i.e., the Gid and its own Sender ID at the time of the original Observe request, as value for the 'request_kid_context' and 'request_kid' elements in the AAD (see Section 4.1.2.3), when decrypting notifications for that observation and verifying their signatures. This ensures that, during the observation's lifetime and across a group rekeying (see Section 4.1.3), the client is able to correctly verify notifications, 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 Public Key* | *Recipient Public Key* | *Shared Secret*
IKM-Recipient = *Recipient Public Key* | *Sender Public Key* | *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.
- The Sender Public Key is the endpoint's own (signature) public key from the Sender Context.
- The Recipient Public Key is the endpoint X's (signature) public key 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' 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]. If EdDSA asymmetric keys are used, the Edward coordinates have to be re-mapped into Montgomery coordinates,

by using the maps defined in [LAN16], before using the X25519 and X448 functions also defined in [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 key 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 multicast request for addressing information. For instance, such a 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, when CoAP Observe is used [HAR15], 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 throughout long-living observations. 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 has to check that the replying server is the expected one, by relying on the server's public key 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.

4.2 *Proxies for CoAP Group Communication*

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*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

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

As explained in Section 3.2, CoAP can be used in group communication environments, e.g., transported over IP multicast. Also in such a case, it is possible to provide end-to-end security, by using the Group OSCORE protocol (see Section 4.1).

In such a group communication scenario, a proxy can be additionally deployed between the origin client and the origin servers in the group. From a high-level point of view, the proxy has to perform the following tasks, similarly to the case of a non-group communication scenario. First, the proxy has to forward a group request from the origin client to the origin servers. Then, the proxy has to relay the possible individual responses from the origin servers back to the origin client.

This setup raises a number of issues, as summarized below.

- In general, after forwarding a group request, the proxy does not know for how long it should accept responses from the origin servers, to be relayed back to the origin client.
- Technically, the origin client sends a single unicast request to the proxy, to be forwarded to the origin servers over multicast. However, the client has to be ready to receive multiple responses to the original group request.
- In general, the client is not able to distinguish the different origin servers producing the different individual responses, or to learn their addressing information for possibly contacting them individually later on.

Furthermore, a number of additional requirements have to also to be fulfilled. That is:

- The proxy has to explicitly identify the origin client. This can rely on a secure communication association used between the client and the proxy, e.g., based on DTLS (see Section 3.4) or on OSCORE (see Sections 3.7 and 4.5).
- Once identified the origin client as above, the proxy has to verify that the client is allowed-listed to perform group requests through the proxy.
- If the origin client and the origin servers protect their communication end-to-end through the proxy, they do so by using Group OSCORE (see Section 4.1).

There is ongoing research work to fully enable the group communication setup discussed above and including a proxy. This has also resulted in the IETF standardization proposal at [TIL21f]. In particular this work introduces a signaling protocol between the origin client and the proxy, which addresses all the issues mentioned above. The following summarizes the main rationale and steps of such a signaling protocol.

When sending the unicast group request to the proxy - to be forwarded to the group of origin servers - the origin client includes in the request a new CoAP option. Its value specifies the amount of time T, in seconds, that the client is fine to wait for multiple responses to the group request. The indication about forwarding the group request – and intended to the proxy – is expressed, as usual, by including in the request the appropriate proxy-related CoAP options. Finally, the client sets an internal timeout, aimed to expire after T seconds.

When receiving the group request from the origin client, the proxy identifies the client and verifies it to be allowed-listed, as discussed above. Then, the proxy: i) sets an internal timeout aimed to expire after T seconds, consistently with the amount of time indicated by the client; ii) consumes the proxy-related CoAP options and forwards the group request to the origin servers over multicast. From then on and until the timeout expiration, the proxy proceeds as follows. When receiving a response to the group request, the proxy adds to the response a new CoAP option. Its value specifies the individual address of the origin server that originated the response. Then, the proxy relays back the response to the origin client.

Until its local timeout expires, the client accepts individual responses to the original group request, as they are relayed back by the proxy. In particular, the client can retrieve from each response the addressing information of the origin server. This allows the client to distinguish the different origin servers producing the different responses, as well as to possibly contact them later on individually, either directly or again through the proxy.

The signaling protocol described above also displays the following benefits.

- It is usable for both classes of CoAP proxies, i.e., forward-proxies and reverse-proxies.
- It is usable also when multiple, consecutive proxies are deployed between the origin client and the origin servers, thus forming a chain of intermediaries.
- It preserves support for cacheability of response messages as originally defined for CoAP. To this end, it adapts both the original response freshness and response validation model. If Group OSCORE is used end-to-end between the origin client and the origin servers, cacheability of responses requires to use additional means, as described in Section 4.4.
- It can be adapted to work in a setup using cross-proxies, especially an HTTP-CoAP proxy. This allows an origin HTTP Client to send a group request to a group of origin CoAP Servers to the proxy. Also in this case, it is still possible to use Group OSCORE to achieve end-to-end security between the origin client and the origin servers, by leveraging the HTTP-to-CoAP and CoAP-to-HTTP mapping originally defined for OSCORE.

4.3 *CoAP Responses over IP Multicast*

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, P-33*
- *Security Requirements: SE-30, SE-32, SE-33, SE-34*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

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

Some use cases display a typical communication pattern where multiple CoAP clients are interested in observing [HAR15] exactly the same resource at a same CoAP server.

A notable example is the Pub-Sub architecture [KOS19], where multiple clients subscribe to a topic, by observing a topic resource at a Pub-Sub Broker acting as server. Upon changes in the topic value, the Broker automatically notifies each subscriber by sending a unicast response. In this particular use case as well as more in general, it would be convenient to have the server sending one single message to all the observer clients, e.g., by using IP multicast. This would obviously result in considerable performance improvements, due to a much lower time in propagating the update, with consequent less utilization of the available bandwidth.

An obvious approach consists in sending such an update as a multicast request to all the observers. However, this would require all the observers to have a dedicated group resource, hence practically acting also as servers. Ideally, a better solution would be distributing the update as a single response message, and as an actual observe notification over IP multicast. However, CoAP currently does not support response messages over IP multicast.

The standardization proposal at [TIL21e] fills this gap and describes how observe notifications can be distributed as responses over IP multicast. In particular, it defines which common Token value should be considered by the server as well as by all the observer clients. Furthermore, it defines how such multicast notifications can possibly be protected by using Group OSCORE.

Intuitively, the approach described in [TIL21e] works as follows. At some point in time, the server can start by itself a "group observation" of one of its own resources. Practically, this may happen when a first client attempts to register as an observer of that resource; or when sufficiently many clients are observing that resource and can all be made group observers.

In either case, the server starts a group observation by crafting and sending to itself a "phantom request" targeting the resource to observe. Such a phantom request has a Token value chosen by the server as available to use. That is, the server is practically in control of the Token space for the group observations, on behalf of the potential observer clients.

While not actually transmitted on the wire, the server sends the phantom request to itself, as if it was sent from a multicast address (which is not practically possible otherwise on a real communication medium). As a result, a group observation is started at the server, as if requested by a group of clients reachable at that multicast address.

Whenever a client tries to normally observe the resource at the server, the latter replies with an error response, with the intent to signal that the client can actually rely on an ongoing group observation. That is, the error response to the client includes a serialization of the phantom observation request. Upon receiving it, the client is able to set itself as taking part of the group observation, for which it uses the phantom request and its Token value as an anchor.

Then, when the observed resource value changes, the server sends a single notification response to the multicast address associated with the phantom request. As per the CoAP protocol, the notification response has the same Token value of the phantom request. Therefore, all the clients listening to the multicast address of the group observation will receive the multicast notification and successfully match it with the phantom request, by means of the Token value.

For a number of reasons, at some point in time the server may decide to cancel a group observation. When this happens, the server simply sends an error response over IP multicast, targeting the registered client and matching the same original phantom request.

As mentioned above, it is possible to protect multicast notifications by using Group OSCORE, having the server and the observer clients as members of the same OSCORE group. To this end, the server first of all protects also the phantom request by using Group OSCORE, and the serialization of such request will again be included in the error response to a new client attempting to observe. Then, each following multicast notification will be also protected with Group OSCORE. In order to ensure that each of such multicast notifications is also cryptographically bound to the phantom request, the Additional Authenticated Data (AAD) used when protecting each of them always includes the 'kid', 'kid context' and 'piv' from the phantom request, as 'request_kid', 'request_kid_context' and 'request_iv' elements (see Section 4.1.2.3).

4.4 *Caching of OSCORE-Protected Responses*

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-31, P-32, P-34*
- *Security Requirements: SE-30, SE-32, SE-33, SE-34, SE-38*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

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

As discussed in Section 3.1, CoAP natively supports the possible deployment of intermediary entities such as proxies. Other than forwarding requests to an origin server on behalf of the origin client as well as relaying back the corresponding responses, the proxy can conveniently cache responses from the server, according to the freshness model defined for CoAP.

Then, if a different origin client sends to the proxy a "similar enough" request to be forwarded to the same origin server, the proxy actually does not need to forward the request. Instead, the proxy can efficiently serve the client, and reply with the response from its cache. In fact, if still valid, such response is what would be anyway obtained by forwarding the request to the origin server.

The above is limited to the caching of responses to requests that perform non-destructive content retrieval operations at a server's resource. With particular reference to CoAP, this is the case for original plaint requests that use the REST methods GET and FETCH.

The same caching principle holds in group communication scenarios, where a proxy can also be deployed between an origin client and the origin servers in the group, as discussed in Section 4.2.

Unfortunately, cacheability of responses at the proxy cannot be seamlessly achieved if end-to-end security is used between the origin client and the origin server(s). This is the case when OSCORE (see Section 3.7) is used for one-to-one communication, or Group OSCORE (see Section 4.1) is used for group communication. That is, the proxy remains capable to cache responses protected with (Group) OSCORE, but it would not be able to use them to serve client requests from its cache.

In fact, different clients that all wish to send a same plain request will practically send to the proxy very different protected requests, thus failing to produce a cache hit at the proxy. As a result, caching of responses becomes pointless in the first place, and the proxy would simply forward every request to the origin server. Clearly, it would be nice if cacheability of responses at the proxy remained possible, also when end-to-end secure communication is used.

To this end, there is ongoing research work to re-enable cacheability of end-to-protected responses. This has also resulted in the IETF standardization proposal at [AMS21b]. In particular, this work builds on Group OSCORE, and introduces the concept of "consensus request", i.e., an end-to-end protected request that all clients are able to identically produce. The following summarizes the main rationale and steps of such a signaling protocol.

When setting up a communication scenario using end-to-end security and cacheability of responses, the following steps are taken.

- An OSCORE group is established. Then, the CoAP clients and servers involved in the communication

scenario join that OSCORE group through its Group Manager (see Section 6.1), and are thus able to communicate in the group using Group OSCORE.

- Upon joining the OSCORE group, all the group members are provided with some additional information, namely: i) a hash algorithm H(); and ii) the OSCORE Sender ID of a Deterministic Client, i.e., a fictitious group member that all the real group members can "impersonate". This information is under the entire control of the Group Manager, and all the real group members can use it to derive the same deterministic keying material.

- A client can prepare a content retrieval request (i.e., a GET or FETCH request) addressed to a server, and then protect it with Group OSCORE by using the keying material associated with the Deterministic Client. While details are omitted in this description, such a request protection relies on an altered version of the pairwise mode of Group OSCORE (see Section 4.1.5). In particular, the request is protected with an encryption key derived from: i) the deterministic keying material; and ii) a hash of the original plain request to protect.

  The interesting point here is that every client in the same group, when wishing to send the same plain request, will thus protect it in the very same way. Therefore, all clients wishing to send a same plain request will produce a same protected "deterministic request", which is in fact a particular instance of the early introduced consensus request.

  Finally, the client includes a new CoAP option in the protected request. The new option fulfils two goals. First, it signals the fact that the request is a deterministic request. Second, it includes as value the hash of the original plain request. This will be later used by the server to enforce the binding between the deterministic request and a response to it.

- The server receiving a deterministic request recognizes it as such and decrypts it similarly to how defined for the pairwise mode of Group OSCORE, with the following differences.
  - The decryption key is derived from the deterministic keying material and from the hash of the original plain request, as retrieved from the received CoAP option.
  - No replay checks are performed on the request, as clarified in the following point.
  - Once produced the decrypted plain request, the server asserts that the request is indeed a GET or FETCH request (i.e., a content retrieval request) and that it is REST-safe to perform on the particular target resource. In fact, in such a case, it is indeed fine to accept such a request, even though: i) it does not have source authentication, since any client in the OSCORE group could have sent it; ii) it might be a replay, since any client in the OSCORE group could have already sent it before.

- After having processed the plain request at the application, the server replies to the client with a response protected with the group mode of Group OSCORE (see Section 4.1.4). Using the group mode is necessary here, to allow the client to verify source authentication of the response. When protecting the response, the server involves in the process also the hash of the original plain request, thus effectively enforcing a cryptographic binding between the response and the corresponding deterministic request. Finally, the server includes in the response a CoAP option Max-Age with value greater than 0, thus indicating the validity time of such response and consistently indicating to the proxy to cache it.

- Upon receiving the response, the proxy will simply cache it, for as long as specified in the conveyed CoAP option Max-Age set by the server.

- Upon receiving the response, the client decrypts and verifies it using the group mode of Group OSCORE. During such a process, similarly to what the server did on its side, the client also involves the hash of the original plain request, thus effectively asserting that the response is cryptographically bound to the previously sent deterministic request.

As mentioned above, if later on a client in the group produces the same plain request, this will result in the same OSCORE-protected deterministic request. Therefore, when reaching the proxy, such deterministic request will produce a cache hit, and the proxy will reply to the client with the response retrieved from its cache, until it is valid and stored. Note that, beyond normal caching capabilities, no additional support is required on the proxy side.

4.5 *OSCORE-capable Proxies*

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-31, P-32*
- *Security Requirements: SE-30, SE-32, SE-33, SE-34, SE-37*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

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

As discussed in Section 3.1, the CoAP protocol natively supports the presence of intermediaries, such us forward- or reverse-proxies. These assist an origin client by performing requests to origin servers on its behalf, and then forwarding back possible related responses.

A number of use cases relying on proxies have the need for a security association also between the origin client and the proxy, to be used for protecting the messages exchanged over that communication leg. In particular, this allows the proxy to securely identify the origin client, before forwarding its request to the origin server. Some of such use cases are summarized below.

CoAP group communication with proxies – CoAP can also be used for group communication, e.g., over IP multicast (see Section 3.2). In particular, communications between a client and the servers in the group can be protected by using Group OSCORE (see Section 4.1).

When a proxy is deployed between the origin client and the servers, it can use the approach described in Section 4.2. That is, following indications from the origin client, the proxy forwards the request to the origin servers in the group, and relays individual responses to the origin client.

In such a case, the proxy has to identify the origin client and verify that it is allowed-listed, before forwarding its request to the group of servers. This raises the need for a secure communication association between the origin client and the proxy.

CoAP observe notifications over multicast – When multiple clients "observe" (see Section 3.1) the same resource at the same server, it is possible for the server to setup a group observation. That is, when the resource representation changes, the server can send a single notification response over IP multicast, thus targeting all the observer clients. This relies on the multicast notifications all matching against a common "phantom request", that the server has provided to all the observer clients as they register their observation (see Section 4.3).

On top of that, if a proxy is deployed and Group OSCORE is used end-to-end between the origin clients and the server, then each client is required to take an additional step, by providing the common "phantom request" to the proxy. Hence, this particular exchange between a client and the proxy is also required to be secured, to especially ensure integrity protection.

External application servers in LwM2M deployments – CoAP is used as message transfer protocol in the OMA Lightweight Machine-to-Machine (LwM2M) standard [OMA-CORE]. Intuitively, a LwM2M Client device can securely "bootstrap" at a Bootstrap Server, and then securely register at a LwM2M Server, with which it performs most of the following communication exchanges. As defined in [OMA-TP], the OSCORE protocol can be used to secure communications also between the LwM2M Client and the LwM2M Server.

In such a case, it is further possible for the LwM2M Client to communicate with an external Application Server, with the two of them also using OSCORE end-to-end. In particular, this would rely on the LwM2M Client using the LwM2M Server as forward proxy, while still using their own original OSCORE Security Context on their communication leg. This allows the LwM2M Server to identify the LwM2M Client, before forwarding its requests outside the LwM2M domain.

While the security association between the origin client and the proxy can generally use the DTLS protocol (see Section 3.4) or other means, it is preferable to rather rely on the OSCORE protocol, especially if (Group) OSCORE is used end-to-end between the origin client and the origin server. However, this clashes with how OSCORE has been originally specified in [SEL19]. That is:

- OSCORE was designed to be used end-to-end only between an origin client and an origin server, i.e., between the actual "application endpoints" as also the only "OSCORE endpoints". Instead, proxies were not supposed to also act as "OSCORE endpoints".

- In its original design, OSCORE does not admit the same CoAP message to be protected multiple times, i.e., by multiple OSCORE layers. However, in the scenarios discussed above, this would need to be the case when OSCORE is used both end-to-end between the origin client and the origin server, as well as between the origin client and the proxy.

There is ongoing research work to fill the gaps discussed above, which has also resulted in a current IETF standardization proposal [TIL21g]. In particular, this work defines how OSCORE can be used also between an "application endpoint" (e.g., an origin client) and a proxy, as well as between two proxies in a chain of intermediaries. That is, a proxy can also be an "OSCORE endpoint." In addition, this work explicitly admits that a same CoAP message can be protected by multiple, nested OSCORE layers applied in sequence, thus yielding an "OSCORE-in-OSCORE" protection.

In order to achieve this, the following deviations from the original OSCORE are introduced.

- When applying an OSCORE layer, some CoAP options that are usually not protected can instead be protected. These include:
    - The OSCORE option, when present in a message as the result of the OSCORE layer immediately previously applied to that same message.
    - A CoAP option intended to be protected for and consumed by the other "OSCORE endpoint" that shares the OSCORE Security Context used to apply the OSCORE layer in question. For instance, if the "OSCORE endpoint" is a proxy, the CoAP option can be one of the proxy-related ones, such as Proxy-Uri, Proxy-Scheme and the Uri-* options.

- When an "application endpoint" applies multiple OSCORE layers in sequence to protect an outgoing message, and it uses an OSCORE Security Context shared with the other "application endpoint", then the first OSCORE layer has to be applied by using that Security Context.
- When receiving a protected message, a recipient "OSCORE endpoint" does not have to treat as an error the case where, after message decryption, the resulting message also includes an inner OSCORE option and thus has to be further decrypted.

Building on the general rules above, the processing of incoming and outgoing messages is also further extended, compared to the original OSCORE specification. In particular:

- When receiving a request, an endpoint assesses which of the following conditions occurs.

  A – The request includes proxy-related options. If the endpoint is actually a proxy, it consumes the proxy-related options and accordingly forwards the request to the (next hop towards the) origin server. Otherwise, it replies with an error.

  B – The request does not include proxy-related options and does not include an OSCORE option. The endpoint is also the intended "application endpoint", thus it delivers the request to its application, if any is present. Otherwise, it replies with an error.

  C – The request does not include proxy-related options, but it includes an OSCORE option. Then, the endpoint decrypts the request by using the OSCORE Security Context retrieved as indicated by the OSCORE option. If decryption fails, the endpoint stops and replies with an error response. After a successful decryption, the endpoint re-assesses which of the three conditions applies to the decrypted request.

  Note that there is no need to introduce additional, explicit signaling information. That is, a recipient endpoint is able to understand what is happening and what to do, based on the (possibly combined) presence of the OSCORE option and proxy-related options.

- When protecting an outgoing response, an OSCORE endpoint applies the same OSCORE layers that have been successfully removed from the corresponding request, but in the reverse order than the one in which they were removed.

- When receiving a response, an OSCORE endpoint expects to remove the same OSCORE layers that it applied in its previous corresponding request, but in the reverse order than the one in which they were applied. However, the recipient endpoint expects to remove at most as many OSCORE layers as it applied to its original corresponding request.

4.6 *Robustness and Resilience against Denial of Service*

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-35, AV-03, AV-04*
- *Security Requirements: SE-24, SE-39*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

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

- *The "Content Distribution Manager" component of the "Secure Communication Layer" module.*
- *The "Network Protection Manager" component of the "Secure Communication Layer" module.*

Maintaining availability of devices in a networked environment is key for effectively providing the intended functionalities of the use case and application. Denial of Service (DoS) attacks ultimately aim to make a targeted device unavailable to other devices attempting to reach it. Practically, such an attack results in impeding legitimate clients from accessing resources on the target server device, thus preventing requests from such clients from being processed and served. DoS attacks can be particularly harmful against IoT networks where many deployed devices can be constrained in terms of energy budget, processing power and network bandwidth. These factors make such devices and the applications they run more susceptible to DoS attacks.

This section presents SARDOS, a security solution for counteracting (Distributed) Denial of Service (DoS) attacks that leverage message flooding and possibly target different communication layers. SARDOS takes a host-based approach, hence running on potential victim servers, and dynamically reacts to the (suspected) ongoing DoS attack in an adaptive fashion, based on the currently perceived attack intensity.

That is, SARDOS dynamically adapts the operative state of the victim server, thus limiting worthless usage or computing/communication resources and energy consumption. At the same time, it also ensures a (best-effort) fulfillment of requests from legitimate clients to the victim server. Furthermore, SARDOS is cross-layer, since it leverages the detection of invalid messages at different communication layers, as well as context-aware, since it regularly considers current attack conditions to accordingly adjust the operational state of the victim server.

Practically, the victim server is able to perform an adaptive and dynamic trade-off between availability and quality of service. That is, as an ongoing attack becomes more severe, the server can gracefully scale down its availability and maintain a reduced but acceptable quality of service for legitimate clients, while also reducing the experienced attack impact. Means to scale down availability and reduce the attack impact include relying on a trusted intermediary to (store-and-)forward messages, as well as temporarily switching off network interfaces, possibly using low-power operating modes to further limit energy consumption.

An early version of SARDOS, together with a proof-of-concept Java implementation [AdaptiveDoS] and a preliminary evaluation was presented in [TIL18], proving that the approach effectively limits the needless usage of resources on a server under DoS attack, while still providing (best-effort) service to legitimate clients.

The current ongoing work extends SARDOS to achieve its goals more effectively in resource- and energy-constrained devices, by also taking advantage of built-in power saving modes. To this end, an implementation for the Contiki-NG operating system is in progress [AdaptiveDoSContiki], as also leveraging multiple OFF states associated with the utilization of low-power modes. This will be used to experimentally evaluate SARDOS on a constrained IoT device under DoS attack, considering different attack intensities.

4.6.1 Invalid Messages as Attack Indicators

Intuitively, SARDOS determines how to adjust the device service behavior by leveraging the intensity of the (suspected) ongoing DoS attack, as inferred from the secure communication protocols already in use.

For instance, incoming protected messages that repeatedly fail to decrypt or to be asserted as non-replayed can be taken as an indication of an ongoing DoS attack. When this happens, the target device can temporarily adapt its performance and service behavior, in order to mitigate the attack effects until the monitored indicators suggest that the situation has become better, i.e., that the attack intensity has fallen below an acceptable threshold.

One example of such secure communication protocols is DTLS [RES12][RES21] (see Section 3.4), which can be used to protect messages exchanged with the CoAP protocol [SHE14] (see section 3.1). However, the same applies also when different secure communication protocols are used, which makes SARDOS independent of the application and able to work within different secure communication stacks.

Along the same lines, SARDOS can also work in concert with non-full-fledged secure communication protocols, such as narrowly scoped security services aimed to deter and detect ongoing DoS attacks. An example is provided by SMACK, a security service that enables early and efficient detection of invalid (DoS) messages [GEH15]. SMACK leverages a short Message Authentication Code (MAC), which is seamlessly embedded in transmitted messages. Then, the message recipient verifies the short MAC and determines if the received message is genuine and coming from a legitimate sender, or is instead invalid and to be discarded.

In [GEH15], SMACK was adapted to work specifically for CoAP messages. That is, the short MAC is embedded in the Token field of the CoAP message header, hence resulting in no communication overhead and no changes to the overall message format. The rest of this section considers this specific adaptation of SMACK to detect DoS attacks, when communications between two CoAP endpoints occur over an insecure channel.

SMACK assumes the presence of a Key Distribution Center (KDC) entity, which is in a trust relation with the recipient device and shares with it a long-term cryptographic key. Upon request, the KDC provides a sender device with: i) a nonce to use as initial Message ID in CoAP messages sent to the recipient device; and ii) unique key material valid for a SMACK session with that recipient device, which expires after a fixed number of messages sent to that device. The sender increments the Message ID after each message transmitted to that recipient in the SMACK session, and uses the session key material to compute the short MAC. By using the Message ID and the long-term key shared with the KDC, the recipient derives the same key material and verifies that the short MAC is correct.

4.6.2 Application Scenario and Adversary Model

The following refers to the scenario shown in Figure 4.1, where a Server S and a Client C communicate using the CoAP protocol (see Section 3.1), either over an insecure channel or using DTLS (see Section 3.4).

Furthermore, we consider an active adversary A that performs a DoS attack against S, i.e., it repeatedly sends invalid CoAP request messages to S, inducing it to parse and process them. We refer to these messages as *attack messages*. This attack induces S to worthlessly commit and use resources, thus endangering responsiveness or even availability altogether for serving requests from legitimate clients. SARDOS aims to reduce this needless processing and waste of resources, while preserving (best-effort) availability.

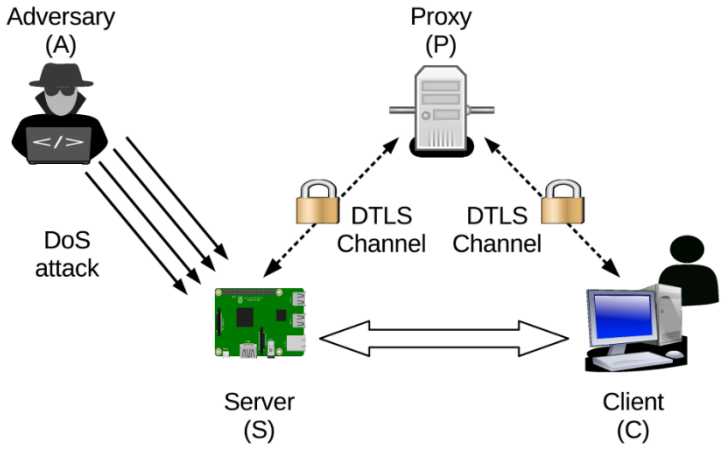

Figure 4.1 - DoS robustness network topology

When under attack, S should be able to identify and discard attack messages. To this end, S can generally use different detection mechanisms, possibly at the same time, in order to distinguish between messages from legitimate clients and invalid messages. For the sake of simplicity and with the intent to later introduce more details about SARDOS, the following focuses on two specific detection mechanisms, i.e., SMACK or the Record protocol of DTLS (see Section 4.6.1). While invalid messages may be due to accidental corruption or processing errors, we refer to a conservative policy where S considers all invalid messages as attack messages.

Also, we consider a Proxy P that, only during DoS attacks, acts as intermediary between C and S. That is, P does not normally participate in the communications between C and S, if S is not under attack and is operating in normal conditions, i.e., it directly communicates with C. The behavior of P is detailed in Section 4.6.3.

A server is associated only to one Proxy, which can however be associated to multiple servers. We assume that both C and S securely communicate with P, by using CoAP over DTLS. If S uses SMACK, then P acts also as KDC (see Section 4.6.1). Also, we consider P trustworthy and designed as reliable and secure, thus practically infeasible to compromise. P can be centralized or based on a distributed architecture. SARDOS is not devoted to any specific architectural design, and further related details are out of scope.

Note that IoT scenarios typically rely on intermediary or third-party entities, as often intended to support asynchronous communication models and to offload effort from server devices. For instance, intermediaries are often used as: i) authorization servers, to enforce access control policies [SEI21][SEI13]; ii) proxies, for message forwarding and caching [SHE14]; iii) key managers, to revoke and distribute cryptographic material.

4.6.3 Reaction against Denial of Service

SARDOS builds on the following rationale: when under DoS attack, it is not convenient for a server to be fully and directly available to serve requests. In fact, resources at the server would be mostly used to handle and process attack messages, thus worsening availability and performance anyway. The server can instead adaptively and gradually enforce a trade-off between direct service availability and protection from DoS. Building on this consideration, SARDOS is designed to achieve two main goals.

- Limiting the impact of DoS on the server's resources. To this end, the server regularly collects the number of received invalid messages, then assesses the attack intensity, and finally adjusts its operative state accordingly. Invalid messages can be detected with multiple, possibly co-existing, security mechanisms at different layers.

- Preserving a (best-effort) capability to serve requests from legitimate clients. To this end, SARDOS relies on the trusted Proxy to assist the server when under attack. The Proxy relays messages between clients and the server during mild/intermittent DoS, and stores client messages to be later forwarded to the server, during intense/persistent DoS.

The following describes the server operative states, the transition among states, and how they affect the client experience.

4.6.3.1 Server Operational Perspective

At any point in time, a server running SARDOS is in one specific operative state from the state machine shown in Figure 4.2, whose evolution is driven by the experienced intensity of DoS attacks.

The three operative states can be described as follows.

- **NORMAL** (negligible/no attack) - When in this state, the server performs as per its *typical* behavior, i.e., it serves client requests at its earliest convenience. Once moved back from the PROTECTED state, the Server notifies the Proxy to be resuming its operations in NORMAL state.
- **PROTECTED** (mild/intermittent attack) - When in this state, the server performs according to a *limited* behavior, i.e., it serves clients only through the Proxy acting as message relay. By doing so, the server reduces its resource consumption when under mild/intermittent DoS attack. Before entering this state, the server notifies the Proxy to be operating in PROTECTED state for t_1 seconds. When specifically returning from the OFF state, the server will receive from the Proxy any stored message.
- **OFF** (intense/persistent attack) - When in this state, the server performs according to a *best-effort* behavior, i.e., it has its network interface turned off and relies on the Proxy to store client requests until it leaves the OFF state. This greatly reduces resource consumption on the server. Before entering this state, the server notifies the Proxy that it will be in OFF state for t_2 seconds, during which the Proxy stores client requests to be relayed and served later.

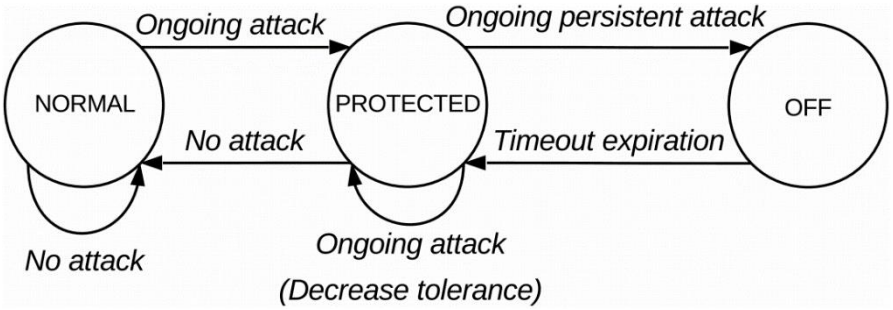

Figure 4.2 - Server state machine

While in NORMAL or PROTECTED state, the server regularly checks the reception rate of invalid messages and determines whether to switch to a different state. To this end, the server considers consecutive time windows, during each of which it maintains a counter X of received invalid messages. If, at the end of a particular time window, the corresponding counter exceeds a set threshold, the server will transition into the next higher operative state. Different threshold values Th_x are used for transitioning into PROTECTED and OFF states, i.e., the attack intensity needs to be more severe for the server to move to the OFF state. Instead, for transitioning

back to a previous lower state, the Server relies on two timers, namely t_1 and t_2, for transitions from PROTECTED to NORMAL and from OFF to PROTECTED, respectively. The threshold values Th_x and the timers t_1 and t_2 are dynamically updated according to the intensity of incoming attack messages, thus appropriately adjusting the attack tolerance. The server always keeps the Proxy informed of which state it is in.

As discussed in Section 4.6.1, invalid messages can be detected at the server through different security mechanisms, which can possibly co-exist at different communication layers. While focus on this respect has been put on using either SMACK or the DTLS Record protocol (see Section 4.6.1), SARDOS is not devoted to any specific detection mechanism, and different approaches than SMACK and DTLS can be seamlessly used, possibly at the same time. In fact, regardless the specific mechanisms and their rationale, each received message assessed as invalid is silently discarded, and results in the counter X incremented by 1. That is, the server enforces a conservative policy where all invalid messages are considered attack messages.

Furthermore, work is ongoing in order to extend the OFF state of the SARDOS state machine, by introducing multiple OFF sub-states as especially relevant for resource- and energy-constrained IoT devices. These sub-states can take advantage of built-in power saving modes, in order to further reduce the energy consumed by the server when in one of the OFF sub-states. Many classes of commercially available constrained devices support low-power modes of operation, which can be controlled via software or hardware. By introducing the OFF sub-states above to reflect varying levels of "depth" for the main OFF state, energy consumption can be further reduced, while still according to the overall dynamic and adaptative rationale of SARDOS. In particular, the OFF sub-states can be mapped to different low-power modes, and the transition to one sub-state or a different one can be chosen as the most appropriate to the current intensity of the ongoing DoS attack. In case of particularly intense attacks, the Server can switch to the most extreme OFF sub-state, hence to the most extreme low-power mode, and shut down most of its functionalities thus maximally reducing its energy consumption.

More investigation is required into how to make use of such extreme operative states in a way which is still overall worthwhile in terms of resource preservation. That is, switching to a specific state at a certain point in time and then back after a certain amount of time might generally risk costing more energy than would be gained in the first place. Hence, it is important to identify good practical criteria for setting and dynamically adjust thresholds and timeout values, such that state transitions effectively occur at the right moment and for an appropriate amount of time, thus ensuring to overall pay off.

This becomes especially relevant when introducing multiple OFF sub-states. For instance, leaving a sub-state OFF_0 in order to move back to the PROTECTED state can require to only switch on the radio interface again. However, moving back to the PROTECTED state from a sub-state OFF_1 or OFF_2 may additionally require making the CPU and other hardware components fully operative again. This in turn results in an additional “wake-up” energy consumption. Thus, it must be ensured by construction that the amount of time spent in a sub-state OFF_1 (OFF_2) always pays off from an energy consumption point of view, compared to rather spending that amount of time in the sub-state OFF_0 (OFF_1). Such an assessment is not trivial to perform, as it must consider the power consumption of the different device components (e.g., CPU, radio interface) as well as the different amounts of energy spent for “waking up” and moving back to the PROTECTED state.

4.6.3.2 Client Operational Perspective

As described below, the client experiences a different service depending on the current state of the server.

- **The server is in NORMAL state** - The client communicates directly with the server as usual. If the client believes the server to be in PROTECTED or OFF state, and sends a request to the Proxy to be

relayed to the server, then the Proxy replies with a SARDOS control message, telling the client to transmit its requests directly to the server.

- **The server is in PROTECTED state** - The server accepts only requests that are relayed by the Proxy, thus the client must send its requests to the Proxy to be relayed to the server. If the client believes the server to be in NORMAL state and attempts to communicate directly with the server, two cases can occur.
    - If the client already has an established communication session with the Server, i.e., through SMACK or DTLS, the server verifies the request to be a valid message, and silently discards it anyway. Then, the server replies to the client with a SARDOS control message, signaling to be currently operating in PROTECTED state.
    - If there is no established communication session, the server does not process the request, counts it as an invalid message (i.e., increments the counter X) and discards it. Then, the server does not reply to the client, which would re-transmit the same request until it reaches its retransmission limit. The network administrator should properly configure the SARDOS parameter on the server side, in order to avoid that relatively few retransmissions from legitimate clients can unfairly induce transitions to OFF state.

  In either case, the client assumes that the server is in PROTECTED state, and starts transmitting to the Proxy all requests intended to the server, indicating to relay them. After that, the client receives the related responses as relayed by P, and assumes the server to be in PROTECTED state until the Proxies notifies that the server has switched to a different state.
- **The server is in OFF state** - The server has its network interface turned off and cannot serve any request. Then:
    - If the client believes the server to be in NORMAL state and contacts the server directly, it does not get a reply and would re-transmit a request until reaching its retransmission limit. After that, the client assumes the server to be in PROTECTED state, and transmits to the Proxy its requests intended to the server, indicating to relay them without storing them.
    - Alternatively, the client might have already known the server to be in PROTECTED state. If so, the client sends to the Proxy its requests intended to the server, indicating to relay them without storing them.

  In either case, the Proxy replies to the client with a SARDOS control message, signaling that the server is in OFF state. Then, the client may still transmit to the Proxy all its requests intended to the server, indicating to store them and to relay them to the server when it switches back to PROTECTED state.
  When doing so, the client can provide an indicative amount of time *t_OFF_C* that it is fine to wait for getting a response from the server, although relayed through the Proxy. Then, the Proxy accordingly stores such requests if the residual time that the server spends in OFF state is less than the amount of time indicated by the client. Otherwise, the proxy discards those request messages.
  In either case, the Proxy informs the client of the residual amount of time to wait until the server switches back to PROTECTED state. This prevents the client from performing further worthless request transmissions during that time interval. Upon receiving responses to requests stored at the Proxy, the client assumes the server to be in PROTECTED and accordingly sends new requests through the Proxy.

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 and Group OSCORE Profiles 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, PE-31*
- *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 [D1.3]:

- *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 briefly overviews two relevant profiles of the ACE framework for authentication and authorization (see Sections 3.8 and 3.8.3), i.e., the OSCORE profile and the Group OSCORE profile.

5.1.1 OSCORE profile

The OSCORE profile of ACE defined at [PAL21] 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).

Upon receiving the 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. 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.

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 they intend to use. Then, C and RS use such values together with the OSCORE Security Context Object received from the AS, to derive a complete, fresh OSCORE Security Context. After that, C sends a first secure request to the RS, protected with the new OSCORE Security Context. Proof-of-possession is achieved when the RS receives such first request and verifies it as cryptographically correct.

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.

5.1.2 Group OSCORE profile

The Group OSCORE profile of ACE defined at [TIL21a] describes how a Client (C) can engage in the ACE workflow to access a resource shared by multiple Resource Servers (RSs) in a group, where communication in the group is protected by using the Group OSCORE security protocol [TIL21b] (see Section 4.1). This requires C as well as all the targeted RSs to have already joined the OSCORE group, for instance using the approach defined in [TIL21c] and also based on the ACE framework (see Section 6.1).

In some applications relying on Group OSCORE, it may be just fine to rely on a "flat" access control model. That is, being a member of the OSCORE group, and hence legitimately owning the group keying material,

automatically grants the right to perform any action at any resource hosted at any group member. While this is acceptable in some scenarios, it is not sufficient for applications that require a more fine-grained enforcement of access control, on a per-node basis.

In particular, it may be necessary to distinguish different classes of clients, e.g., low-privileged or high-privileged, so that some group members can perform only some operations on some resources hosted at other group members. One can achieve this by creating multiple OSCORE groups, i.e., one for each class of access rights that clients belong to. However, this would require additional, inconvenient key management operations, and especially revocation and renewal of the keying material of all the involved groups, if access rights of any group member change.

The Group OSCORE profile addresses this type of requirements, and enables fine-grained access control using ACE. In particular, it does not require the creation of additional OSCORE groups than the ones already set up and deployed. Furthermore, it deals with the fact that no other profile of ACE supports secure group Communication.

Assuming C and the RSs as already members of an OSCORE group, C sends a Token request to the AS, specifying also: i) the Group ID of the group; ii) the identifier it has in the group; iii) the public key it uses in the group; and iv) a Proof-of-Possession (PoP) evidence (e.g., a signature computed using its own private key), over a challenge that the AS is also able to derive, as tight to the secure communication channel it has with C.

Then, upon receiving the Token request from C, the AS verifies the PoP evidence, and creates an Access Token which includes the Group ID, the client identifier and the public key received from C. After that, the AS provides C with the Access Token. For the sake of proof-of-possession, C has to prove to the RS for which the Access Token has been issued to also possess the private key corresponding to the public key specified in the Access Token.

After that, C uploads the Access Token to the correct RSs in the group. Upon receiving the Access Token from C, the RSs can retrieve the public key of C in the group, and verify with the OSCORE Group Manager that C is indeed a member of that group and owner of that public key.

Note that C can acquire and upload one Access Token for each RS in the group, or instead a single Access Token intended for multiple RSs at once, if the AS supports its release.

Finally, C can send (multicast) messages to the group protected with Group OSCORE, reaching all the RSs in the group. At a given RS, proof-of-possession is achieved when the RS receives such first request and verifies it as cryptographically correct, by using the public key of the client. This includes verifying the signature of the message (if using the group mode of Group OSCORE), or verifying the integrity of the message using a pairwise key derived from the asymmetric keys of both C and RS (if using the pairwise mode of Group OSCORE).

5.2 *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*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

- *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) eventually expire, and the AS can give an explicit indication of expiration time. When that happens, both a Client (C) and a Resource Server (RS) owning that Access Token would discard it. In particular, C would have to get a new Access Token from the AS and upload it to the RS, before continuing accessing resources at that RS.

On top of that, there are additional circumstances when an Access Token may be revoked, 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 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), a 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. 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 specified in the standardization proposal [TIL21d] aims to fill this gap. That is, it specifies a method for registered devices to access and observe a Token Revocation List (TRL) resource on the AS, in order to get 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 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, but rather 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 alternatively 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 the TRL resource.

More specifically, a registered device can access the TRL resource at the AS in different modes, which result in different responses from the AS.

- Full query mode: the AS returns the token hashes of the revoked Access Tokens currently in the TRL and pertaining to the registered device that has sent the request.

- Diff query mode: the AS returns a set of diff entries. Each entry is related to one of the N most recent updates in the portion of the TRL pertaining to the registered device that has sent the request, where N is specified as a query parameter of the request. In particular, the entry associated with one of such updates contains a list of token hashes, such that i) the corresponding revoked Access Tokens pertain to the issuer of the request; and ii) they were added to or removed from the TRL at that update. This mode of operation can potentially be extended, in order to allow a registered device to retrieve a set of diff entries not only as limited to the most recent TRL updates, but rather starting from an arbitrary point in time taken as resumption point.

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 [D1.3]:

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

The *Usage Control System* (UCS) regulates the usage of a resource following the UCON model [PAR04] (see Section 3.9). The UCS 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 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 a mutable attribute changes its value, 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 change.

The UCS is the core of the UCON framework, which includes other components that interact with the UCS. Figure 5.1 shows the UCON framework architecture and its main components, which are described in the following.

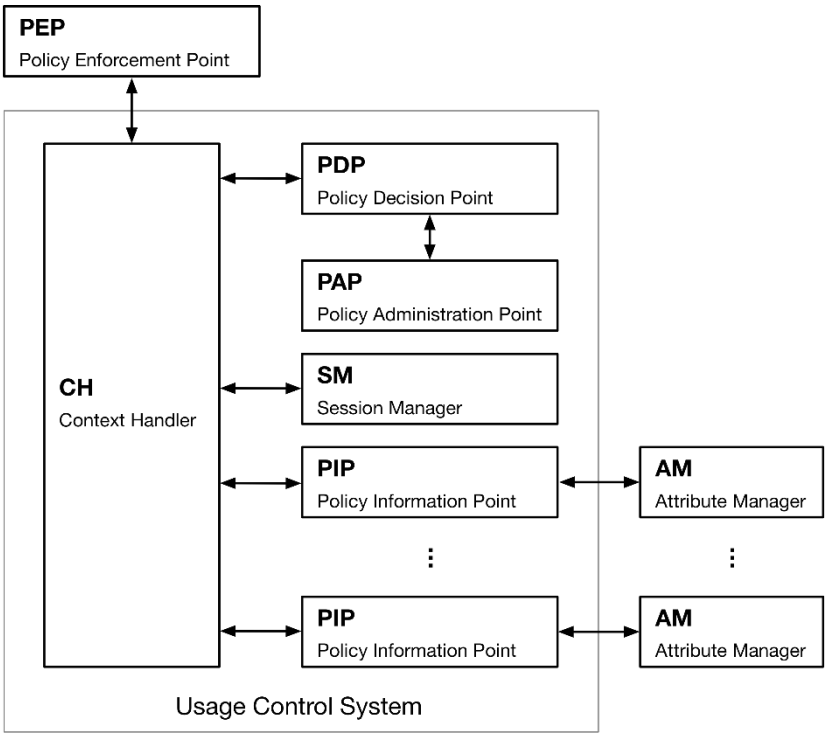

Figure 5.1 - UCON framework architecture

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 and interacts with the UCS through three different types of requests: *tryAccess*, *startAccess*, and *endAccess*.

- **tryAccess** request: 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 an XACML request (*access request*) to the UCS for evaluation (step 1 in Figure 5.2). More in detail, the request is sent to 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 5.2. 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** request: upon receiving a permitAccess response following a tryAccess request, the PEP sends a startAccess request 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 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 5.2. If a negative response is received, the PEP terminates the access to the resource *r* for the subject *s* and sends an endAccess request to the CH.

- **endAccess** request: the PEP sends this request to the UCS as soon as the access to the resource terminates. An access can be terminated for two different reasons: (i) the access has naturally ended (step 5(i) of Figure 5.2), or (ii) the PEP received a revokeAccess message from the UCS (step 5(ii) of Figure 5.2). 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 request (step 6(ii) of Figure 5.2).

  In both scenarios, the PDP evaluates the original 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 5.2. Note that, independently of the UCS response, the access to the resource terminated before the endAccess request was sent by the PEP.

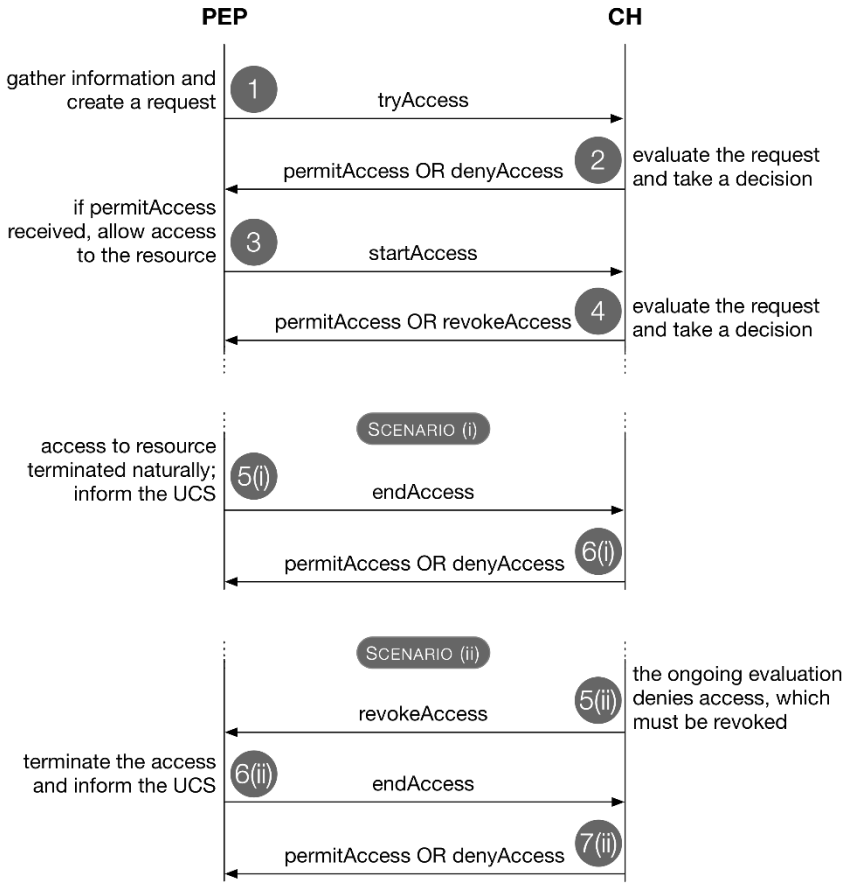

Figure 5.2 - PEP-CH interaction

The **Policy Administration Point (PAP)** component manages and stores the UCPs. Each UCP must be written in 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 request, 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 evaluating the access request.

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. Although less useful, 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 commands the AM 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 5.2 and

coordinates the process of evaluation of access requests.

For any type of access request and for performing a policy re-evaluation on it, 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 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 to the CH, which undertakes different actions depending on the type of request it is serving:

tryAccess request:

- 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 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 request:

- 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 request:

- 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 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 request is Permit; it updates the session after the evaluation of a startAccess request or if the access decision following a policy re-evaluation is Deny; and it deletes the session after the evaluation of an endAccess request.

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.

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 [D1.3]:

- *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 a revocation of issued Access Token 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.

The work-in-progress Java implementation from CNR [ACE-UCON-DEV] has been integrating 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.

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 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 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 5.3.

```
<?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 5.3 - 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. 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. 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 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 on which access to be granted. Finally, the /token endpoint expresses the resources through a scope and includes it in the response returned to the Client.

As an example (also shown in Figure 5.4) the scope SCOPE could refer to the set of resources {RES1, RES2}, the scope SCOPE1 to {RES1}, and the scope SCOPE2 to {RES2}. The operation on all the resources is assumed to be READ. Let a client C specify SCOPE as scope and AUD as audience in its Access Token request. The AS obtains the set of resources corresponding to the scope SCOPE, i.e., {RES1, RES2}, and the related operation, i.e., READ. Then, the PEP creates the XACML requests and individually submits them to the UCS. Each request contains C as subject, AUD as audience identifying the resource server, READ as action, and either RES1 or RES2 as resource. Then, the UCS evaluates the two requests and returns an access decision for each request. Let the access decision for RES1 be Deny and the access decision for RES2 be Permit. Then, the AS expresses {RES2} through the scope SCOPE2 and includes such a scope in its response to the Client.

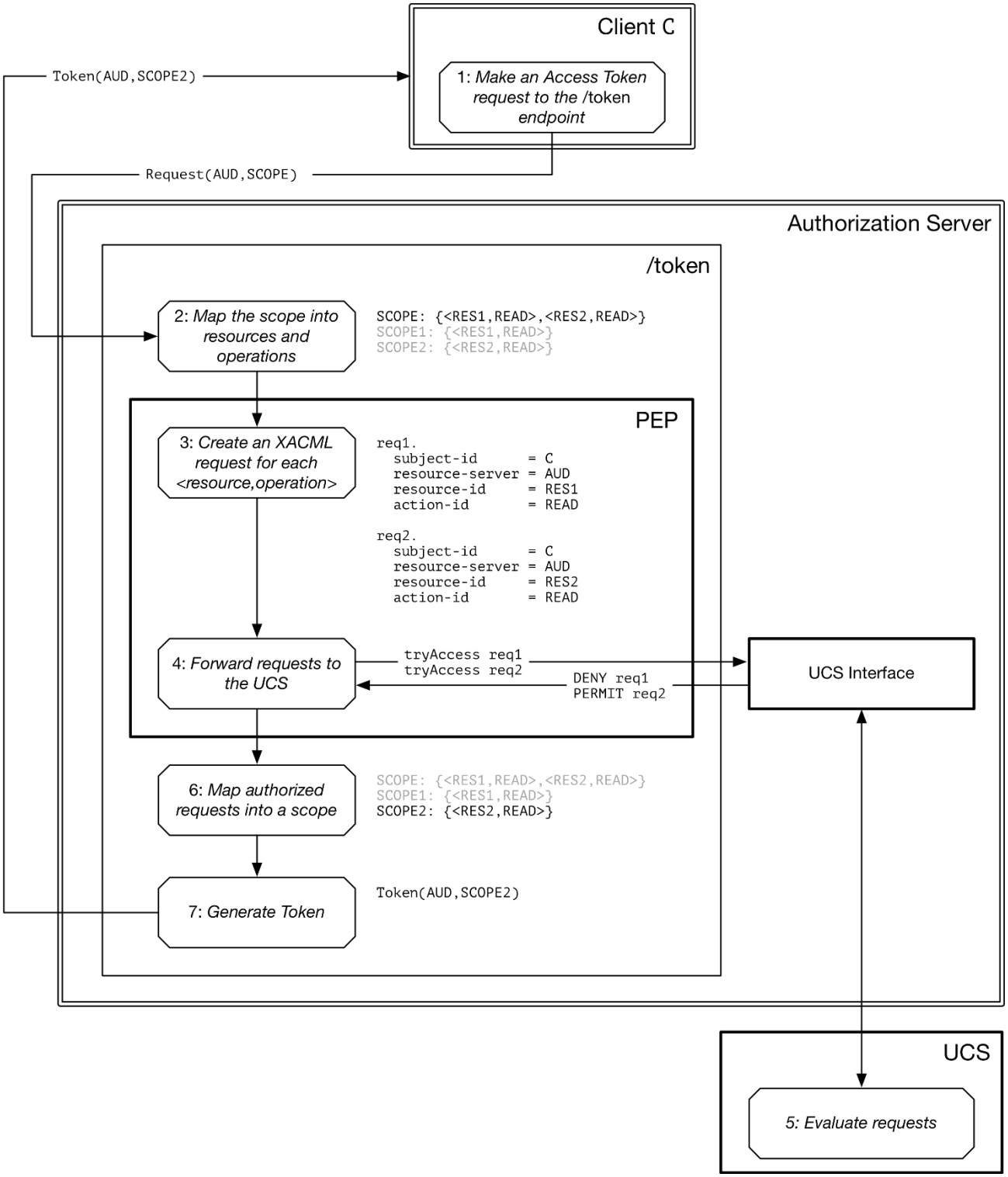

Figure 5.4 - Diagram showing how the UCON framework has been integrated into the ACE framework.

5.4.2 From access revocation to Access Token revocation

The UCS 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 accesses to single resources. On the other hand, the ACE framework considers the possible revocation of Access Tokens as a whole. Since a single Access Token can grant access to more than one resource, 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 at a 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 to be 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 request to the UCS. Both these messages convey information about the session to be revoked, i.e., the SessionID.

In order to correctly integrate the UCON framework into ACE, 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 request to the UCS for each session associated with the Access Token. Also, 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 by means of a SIFIS-Home application. They 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 an access policy for the tenants and the resource "washing machine high-temp cycle", and it adds the attribute "daily energy consumption" in the on- section of the UCP, specifying that its value must be lower than a certain threshold in order to let tenants perform high-temperature wash cycles. The daily energy consumption value is managed by an AM, which is the smart meter deployed in the smart home.

Within the SIFIS-Home application, a tenant selects the high-temperature wash cycle and then starts the washing process. If an Access Token has not already been issued to the user 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 washing machine high-temp cycle is sent to the AS. The access is granted by the UCS, and the Access Token is issued to the tenant, which can now trigger the execution of the high-temperature wash cycle.

In case the daily consumption threshold is passed, the Access Token is revoked. That is, the policy re-evaluation at the UCS returns an access decision of Deny because the current daily energy consumption value is higher than the threshold value set within the policy. Then, the UCS sends a revokeAccess message to the PEP, which replies with an endAccess request. Then, the token hash of the revoked Access Token is stored in the TRL of the AS, and the notification procedure from Section 5.2 is performed.

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

6 Part 3 – Establishment and Management of Keying Material

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

6.1 *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 [D1.3]:

- *The "Authentication Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Key Manager" component of the "Secure Lifecycle Manager" module.*
- *The "Device Registration 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 [TIL21a] 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 by using the ACE framework for authentication and authorization [SEI21] introduced in Section 3.8. In particular, [TIL21b] is a standardization proposal which defines how 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.

A Java implementation of the above services from RISE is available at [ACE-DEV], as integrated in a Java implementation of the overall ACE framework and available for use in the SIFIS-Home project.

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

- A joining node acts as ACE client, thus requesting an Access Token from an Authorization Server, 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.

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. [PAL21a][GER21], 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 a 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.
2. The joining node obtains the Access Token and uploads it at the Group Manager, which validates it and stores it.
3. The joining node sends a joining request to the Group Manager, targeting the group-membership resource associated with the OSCORE group. The joining node specifies in the joining request:
   a. The name of the exact OSCORE group it wants to join, and the particular role(s) it wants to take in the group.
   b. Its own public key, corresponding to its own private key to use in the group.
   c. A Proof-of-Possession (PoP) of its own private key, in order to prove possession of such key to the Group Manager. The PoP input is a challenge that both the joining node and the Group Manager can build, possibly using information 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. If the OSCORE group uses only the pairwise mode, the PoP evidence is a MAC computed with 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 asymmetric keys.
   d. Optionally, an indication of interest to retrieve the public keys of other 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 joining request, 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 communicate in the group by using Group OSCORE. This especially includes the Master Secret, the Context ID, and a Sender ID uniquely assigned to the joining node.
   b. The Group Manager's public key, together with a PoP evidence of its own private key and a nonce used as PoP input to compute the PoP evidence. The PoP evidence is computed according to the same approach used by the joining node to compute its own PoP evidence included in the Join Request.
   c. If requested, the public keys of the other group members.
   d. Optionally, the list of communication policies adopted in the group.

Later as an active group member, a node can further interact with the Group Manager, according to a dedicated RESTful interface. In particular, a current group member can perform the following operations at the Group Manager, by targeting the group-membership resource, or some of its dedicated sub-resources, associated with the OSCORE group at hand.

- Request for the current group keying material.
- Request for a new Sender ID, e.g., in case of exhausted Sender Sequence Number space.
- Request for the public key of the Group Manager.
- Request for the public keys of the current group members, or of a selected subset.
- Request for the set of "stale" Sender IDs previously associated with group members and recently relinquished, i.e., due to a requested change or to having left the group.
- Request for the current communication policies in the OSCORE group.
- Request for the current version of the group keying material.
- Request for the current status of the OSCORE 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 own public key, which replaces the current one.
- Leave the OSCORE group.

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 [TIL21b] 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 Recipient Contexts and stored public keys. 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 OSCORE group.

6.1.1 Discovery of OSCORE Groups

After deployment, a CoAP endpoint may be in possession of only a limited amount of operative information. In particular, the endpoint may have been programmed/provided with the names of an OSCORE group to join through the respective Group Manager (see Sections 4.1 and 6.1). However, the endpoint may not know in advance other information required to do that, especially the URL of the group-membership resource at the Group Manager to join the group.

That is, the endpoint faces the problem of discovering the OSCORE group, i.e., of discovering the group-membership resource at the Group Manager to join it. This practically means discovering the link to such resource. The standardization proposal at [TIL21d] describes how this can be achieved, by using an already well-known approach to discover links to CoAP resources, i.e., the CoRE Resource Directory (RD) [AMS21a].

Intuitively, a CoAP server can register itself at the RD, and then register multiple entries, i.e., one for each of its own resources. Each registered entry includes a link to the corresponding resource, possibly together with target attributes describing the resource and the link itself. Later on, a CoAP client can perform a resource lookup at the RD, possibly by filter criteria, in order to retrieve links and target attributes related to registered resources.

Thus, the approach described in [TIL21d] builds on the RD as follows:

- The OSCORE Group Manager registers itself with the RD, and registers the links to the group-membership resources corresponding to its own OSCORE groups.
- A CoAP endpoint that wishes to join an OSCORE group, can perform a resource lookup at the RD, in order to retrieve the link to the right group-membership resource of the Group Manager, where to send a joining request (see Section 6.1).

This approach displays the following side features and benefits.

- The CoAP endpoint can query for the link to join OSCORE groups by using a number of different lookup criteria. This makes it possible to discover the OSCORE groups used by different applications sharing group resources, as separately registered with the RD.

- Target attributes of discovered links can be used to early provide additional information related to the OSCORE group at hand. This includes especially the list of algorithms used in the OSCORE group and the URL to the ACE Authorization Server associated with the Group Manager. Getting knowledge of this information early in time spares the CoAP endpoint to engage in later additional exchanges with the Group Manager, and to early understand if it supports the current configuration of the OSCORE group altogether.
- The CoAP endpoint may observe [HAR15] the entries at the RD, thus getting automatic notifications about the link for a particular OSCORE group. This has two advantages. First, it automatically informs about possible changes in the link to the group-membership resource at the Group Manager, and about how the OSCORE group itself currently works (through the target attributes of the link). Second, it addresses a corner case where the CoAP endpoint is deployed before the OSCORE group has been created or even before the Group Manager has been deployed. In such cases, an automatic notification will reach the observer CoAP endpoint, once the OSCORE group is actually existing and available to be joined.

6.2 *Configuration of OSCORE Groups using ACE*

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

- *Functional Requirements: F-54, F-55*
- *Non-Functional Requirements: PE-28, PE-29, PE-31*
- *Security Requirements: SE-30, SE-35, SE-36*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

- *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.*

The OSCORE Group Manager (see Section 4.1.1) can provide an additional RESTful interface intended for an Administrator user, rather than for candidate and current members of an OSCORE group. This is defined in the recent standardization proposal [TIL21c].

This interaction with the Group Manager also builds on the ACE framework for authentication and authorization [SEI21] (see Section 3.8), i.e., the Administrator acts as ACE client and the Group Manager acts as ACE Resource Server. That is, the Administrator has to obtain an Access Token from an authorization Server and uploads it at the Group Manager, in order to prove to be authorized to perform administrative operations through the admin interface.

The admin interface at the Group Manager is organized as follows:

- A single group-collection resource acts as root resource, and represents the collection of the configuration of all the existing OSCORE groups under the Group Manager.
- Several group-configuration resources, one for each existing OSCORE group. A group-configuration resource contains the configuration of the corresponding OSCORE group, which is structured as follows:
    - The "configuration properties" describe how the OSCORE group works, i.e., the used cryptographic algorithms and related parameters.
    - The "status properties" describe additional information about the corresponding OSCORE group. This includes: the group name and description, the current group status and group policies, the URL to the corresponding group-membership resource to join the group (see Section 6.1),

and the URL to the ACE Authorization Server associated with the Group Manager.

The Administrator can perform the following operations at the admin interface of the Group Manager, distinctly by interacting with the group-collection resource or one of the group-configuration resources.

The available operations on the group-collection resource are:

- Retrieval of the list of current group-configuration resources, i.e., a list of links to those, possibly by applying filter criteria.
- Creation of a new OSCORE group, i.e., creation of a new configuration resource for that group. When doing so, an initial configuration content can be provided, otherwise the Group Manager considers default values. Practically, the Group Manager creates both a group-configuration resource and a group-membership resource associated with the group.

The available operations on a group-configuration resource are:

- Complete retrieval of the current configuration of an OSCORE group.
- Partial retrieval of the current configuration of an OSCORE group, by applying filter criteria.
- Complete update (i.e., total overwriting) of the configuration of an OSCORE group.
- Selective update of the configuration of an OSCORE group (i.e., only some parameters).
- Deletion of an OSCORE group, i.e., deletion of the corresponding group-configuration resource and group-membership resource.

6.3 *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 [D1.3]:

- *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.*

The Ephemeral Diffie-Hellman over COSE (EDHOC) [SEL21] protocol is a very compact and lightweight authentication protocol for performing a security handshake and establishing a cryptographic secret between two peers. In particular, EDHOC has as a main use case the establishment of a Security Context, that the two peers can use to protect their communication 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 [SCH17] for cryptography (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.

Authentication of the two peers can be based on raw public keys (RPKs) or public key certificates, and it requires the application to provide input on how to verify that the endpoints to authenticate are trusted. The authentication credentials including public keys can be conveniently identified by reference also using COSE, thus preserving a low communication overhead during the EDHOC execution. Also, different types of authentication credentials are supported, e.g., CBOR Web Token (CWTs) / CWT Claims Sets (CCSs) [JON18], X.509 certificates [BOE08] and CBOR encoded X.509 (C509) certificates [MAT22].

Orthogonally to the above, EDHOC provides authentication according to 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 static Diffie-Hellman public keys, which in turn is used to compute a MAC as authentication evidence. Clearly, the latter method yields EDHOC messages that are smaller in size.

As to its core key establishment process, EDHOC makes use of known protocol constructions, such as the SIGMA protocol [SIGMA] 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 IoT.

In short, the two EDHOC peers are denoted as Initiator (i.e., the sender of the first EDHOC message) and Responder. After having successfully exchanged three EDHOC messages, both peers agree on a same cryptographic secret, which they can use to derive further, specific security material. After that, the Responder may send an optional fourth message to make the Initiator promptly achieve key confirmation, e.g., in scenarios where the Responder never sends protected application messages to the Initiator.

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.

Figure 6.1 shows the EDHOC message exchange, abstracting from the specifically used cipher suite and transport protocol. Examples of message sizes for EDHOC is given in Section 1.3 of [SEL21]. More details on the design, features, properties and current status of the EDHOC protocol are provided in [VUC22].

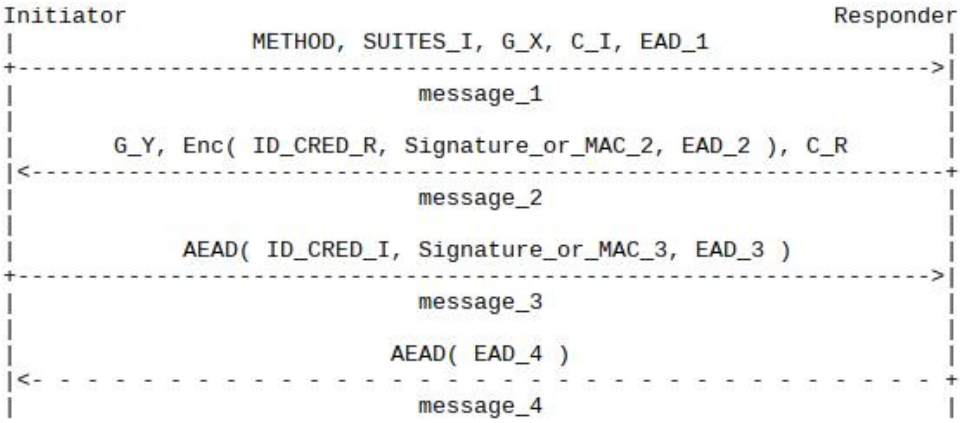

Figure 6.1 - Execution of the EDHOC protocol, including the optional fourth message

An implementation of the EDHOC protocol from RISE for the Californium library [CALIFORNIUM] from the Eclipse Foundation is available at [EDHOC-DEV].

6.3.1 Profiling EDHOC for CoAP and OSCORE

While generally transport-independent, EDHOC messages can be specifically transported as payload of CoAP messages. Also, while the establishment of a cryptographic secret can have general applicability, the main use case for EDHOC is the establishment of an OSCORE Security Context (see Section 3.7.1), as derived from the secret established through an EDHOC execution.

An EDHOC execution requires the two peers to exchange three EDHOC messages. With reference to the CoAP protocol, the following assumes a CoAP client to take the role of EDHOC Initiator (thus sending the first EDHOC message) and a CoAP server to take the role of EDHOC Responder.

As shown in Figure 6.2, the typical 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. After that, the client and server have authenticated one another and agree on a cryptographic secret, from which they can derive an OSCORE Security Context. Then, the client and server can exchange further CoAP messages protected with OSCORE.

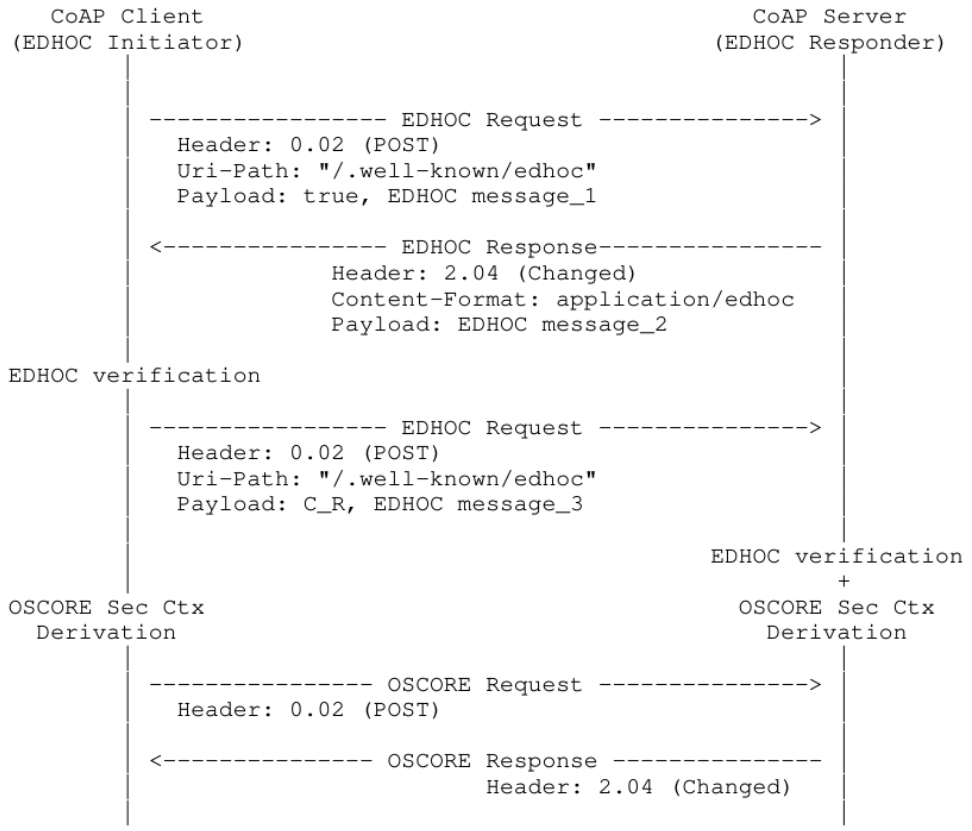

Figure 6.2 - Execution of the EDHOC protocol over CoAP

There is ongoing research work for specifically “profiling” 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 [PAL21c], 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 actually eligible for an 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 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.

This is practically achieved as defined in [PAL21c] and shown 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.

In particular, such a request includes a new “EDHOC” CoAP option, which signals that the message is a combined request and that its payload conveys both EDHOC message_3 and the actual data intended for the application request. Except for EDHOC message_3 itself, the client protects this combined request using the established OSCORE Security Context.

When receiving the combined request, the server notices the signaling EDHOC option and thus extracts EDHOC message_3 from the request payload. Then, the server performs the same operations as if it had received a stand-alone EDHOC message_3. After that, the server has also completed the EDHOC execution and derived the OSCORE Security Context shared with the client. Thus, the server is able to use that OSCORE Security Context and successfully decrypt the received combined request to finally retrieve the intended application data.

By combining EDHOC message_3 with the first protected application request into a single message, 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, which is however the typical and default case.

The optimized workflow 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.

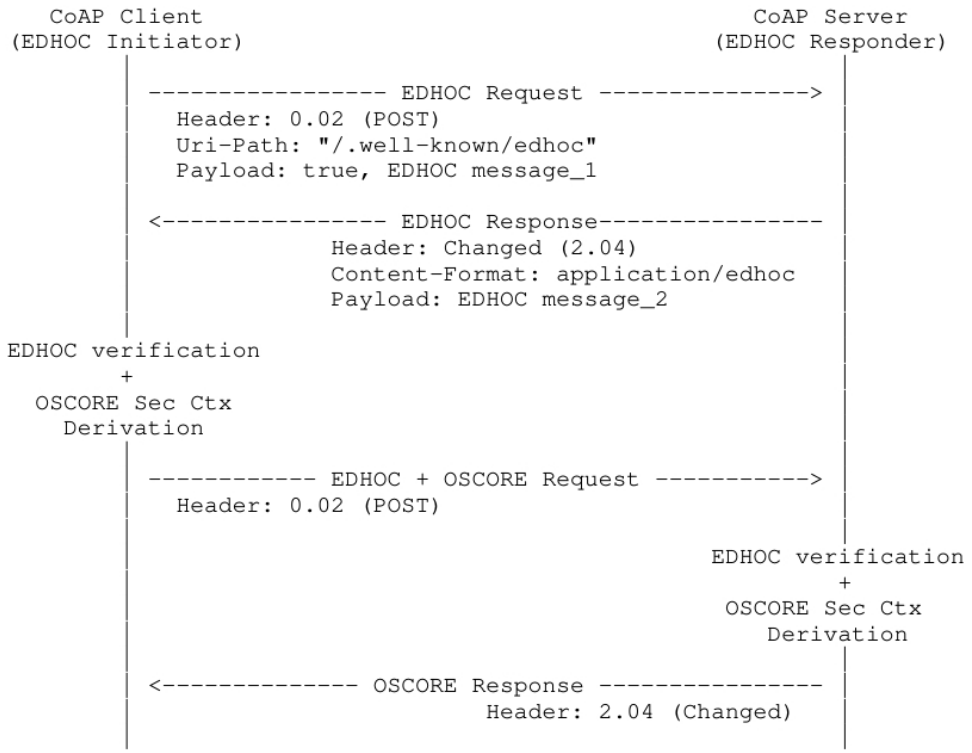

Figure 6.3 - Optimized execution of the EDHOC protocol over CoAP

Conversion of EDHOC identifiers to OSCORE identifiers. Using EDHOC to establish an OSCORE Security Context involves two types of identifiers. On 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).

The EDHOC specification [SEL21] defines a method for converting the EDHOC connection identifiers used in an EDHOC execution to corresponding 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. In particular, this method maps either of two equivalent EDHOC connection identifiers into a same OSCORE Sender/Recipient ID.

Instead, this profiling contribution defines the reverse identifier mapping, i.e., how an OSCORE Sender/Recipient ID is deterministically converted to exactly one of the two corresponding equivalent EDHOC connection identifiers. This approach is necessary to use in case the optimized workflow discussed above is supported by the

CoAP server and thus possibly used during an EDHOC execution. More generally, even when not strictly necessary to use, this approach is preferable. In fact, it always converts an OSCORE Sender/Recipient ID to exactly the EDHOC connection identifier with the smallest size among the two corresponding equivalent ones.

The optimized conversion of OSCORE Sender/Recipient IDs to EDHOC connection identifiers 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.

Web linking to facilitate discovery of EDHOC resources. As explained above, an EDHOC execution involves the interaction with an EDHOC resource at a server. One or multiple EDHOC resources at the same server can be associated with an "applicability statement". This includes a number of information elements describing how EDHOC can be run with the server, when interacting with any of the 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 [AMS21a]. 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 profiling contribution defines a number of parameters, each of which corresponds to different information elements possibly present in an EDHOC applicability statement. In particular, these parameters can be used as target attributes accompanying a discovered link to an EDHOC resource, consistently with the CoRE link-format.

Thus, a CoAP client that discovers the link to an EDHOC resource can at the same time learn about the applicability statement 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.

6.4 *Key Usage Limits and Lightweight Key Update 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*
- *Security Requirements: SE-30, SE-31, SE-43, SE-46, SE-47*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

- *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.*

As described in Section 3.7, the OSCORE security protocol uses AEAD algorithms to encrypt and integrity-protect exchanged messages. In particular, two peers using OSCORE rely on the Sender/Recipient keys included in their shared OSCORE Security Context, in order to protect the messages exchanged with one another.

However, AEAD algorithms intrinsically display limits in the secure usage of keys. That is, an adversary starts having a statistical "advantage" in breaking the security properties of the used AEAD cipher in case: i) a key is used for encryption operations more than a certain number of times; or ii) a key is used for decryption operations after a certain number of failed decryptions with that key has occurred. A theoretical framework for computing these limits is available at [GÜN21]. In addition, a key may simply expire as naturally comes to the end of its validity time.

In order to address the issues above, two OSCORE peers have to renew their Sender/Recipient keys when approaching the associated usage limit or expiration time. While there are some different approaches for two peers to establish a new OSCORE Security Context, those have some disadvantages, or are inconvenient for *updating* an already established Security Context.

There is ongoing research work to better understand and more efficiently address the issues above with particular reference to OSCORE. This activity is also a current IETF standardization proposal [HÖG21], and can be broken down into two contributions.

Recommended limits of key usage. Although still taking the analysis at [GÜN21] as a starting point, the work documented in [HÖG21] has been defining ideal key usage limits to adopt when AEAD algorithms are used specifically in OSCORE. For different possible AEAD algorithms, this specialized analysis has been defining ideal triples of values (q, v, l) that can be considered by two OSCORE peers as limits to comply with. In particular, 'q' indicates the maximum number of acceptable encryptions with a same key; 'v' indicates the maximum amount of permitted failed decryptions with a same key; and 'l' indicates the maximum size of data protected at each encryption, expressed in cipher blocks. The challenge is about defining the limits above in such a way that, by complying with them, it is ensured that an adversary does not gain any significant statistical advantage towards breaking the security properties of the used AEAD algorithm.

Key update procedure (KUDOS). As discussed above, approaching the key usage limits requires two OSCORE peers to renew their Security Context and related keying material. However, current methods have disadvantages or are not really efficient for updating an already existing Security Context. Therefore, the work documented in [HÖG21] is defining a new and efficient key update procedure for OSCORE, namely KUDOS. This is loosely based on the procedure defined in Appendix B.2 of [SEL19], while overcoming its drawbacks and providing several advantages.

In particular, KUDOS allows two OSCORE peers to update their current OSCORE Security Context in a lightweight and efficient way and displays the following desirable properties.

- KUDOS can be initiated by either of the two OSCORE peers.
- Once completed the execution of KUDOS, the new OSCORE Security Context enjoys Forward Secrecy.
- The new OSCORE Security Context preserves the same ID Context value of the old OSCORE Security Context. Such value does not change during the execution of KUDOS.
- KUDOS is secure to use also in case either of the two peers reboots during its execution.
- KUDOS is completed in only one round-trip, after which both peers share the new OSCORE Security Context. After that, the two peers achieve mutual proof-of-possession in the immediately following message exchange, which is protected with the new OSCORE Security Context.

A KUDOS message is signaled to be processed as such by means of the CoAP OSCORE option. To this end, the OSCORE option is extended to use a newly defined flag bit 'd'. When such a flag bit is present and set, the message in question is specifically a KUDOS message, and its OSCORE option additionally transports a new field 'id detail'.

From a high-level point of view, KUDOS consists in the two peers exchanging two random values R1 and R2, that together compose a nonce N. The two values R1 and R2 are transported in the 'id detail' field of the OSCORE option of KUDOS messages.

Then, the two peers can use the nonce N to practically derive a new OSCORE Master Secret and OSCORE

Master Salt, from which a completed OSCORE Security Context can be in turned derived. The exact use of the nonce N can rely on two different approaches, depending on whether the two peers established their "original" OSCORE Security Context by means of the EDHOC key establishment protocol [SEL21] or not. In the former case, the nonce N is conveniently provided to the *EDHOC-KeyUpdate()* method available in EDHOC implementations. In the latter case, the nonce N is used with the standard HKDF-Expand() method [KRA10]. The two alternatives are conveniently abstracted by the KUDOS interface method *updateCtx()*.

Figure 6.4 shows the KUDOS execution, considering the client-initiated workflow.

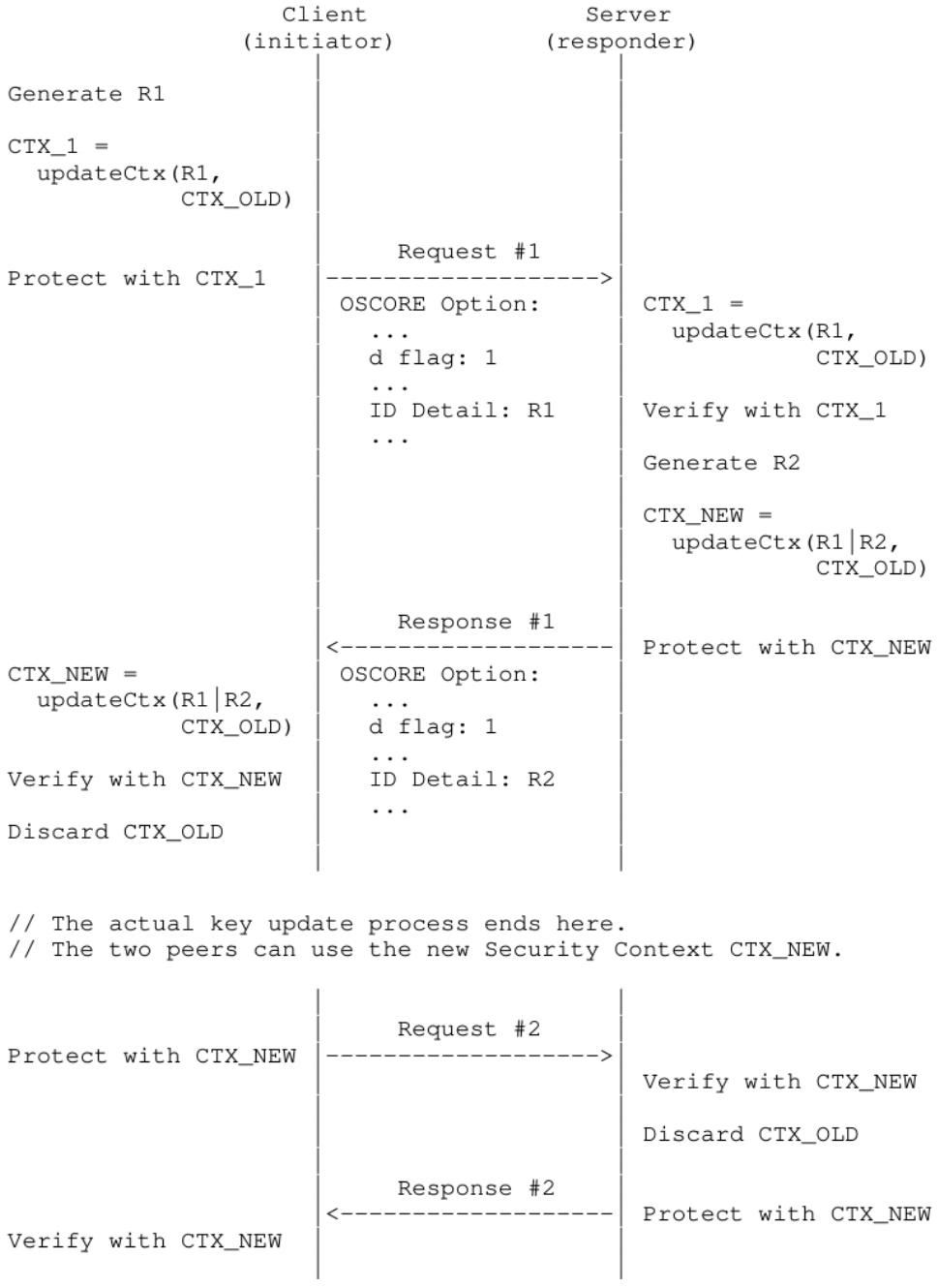

Figure 6.4 - Client-initiated KUDOS workflow

6.5 *OSCORE and EDHOC in the OMA LwM2M Management Framework*

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-31, SE-35, SE-43*

The solutions and methods presented in this section pertain to the following components defined in [D1.3]:

- *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.*

The OMA standard "Lightweight Machine-to-Machine" (LwM2M) provides a control and management framework for IoT devices [OMA-CORE]. The framework builds on an architectural model including a LwM2M client, a Bootstrap Server, as well as a LwM2M Server acting as Device Manager.

In principle, an IoT device acting as LwM2M Client first "bootstraps" at the Bootstrap Server over a (typically pre-established) secure association. As a result of the bootstrapping process, the LwM2M Client receives also parameters and security material to use for securely "registering" at the LwM2M Server. Once completed the registration process, the LwM2M Client and Server can securely communicate with one another, typically for retrieving information from or issuing commands to the LwM2M Client. To this end, the standard provides a data model with an extensible set of "LwM2M Objects", together with different available encoding.

Besides the architectural model, workflow and LwM2M Objects, the standard defines a number of transport bindings for practically and securely exchanging LwM2M messages among the parties involved [OMA-TP]. In particular, the CoAP protocol (see Section 3.1) is typically used for message delivery. Also, communications can be secured in different ways, e.g., end-to-end by using the OSCORE security protocol (see Section 3.7).

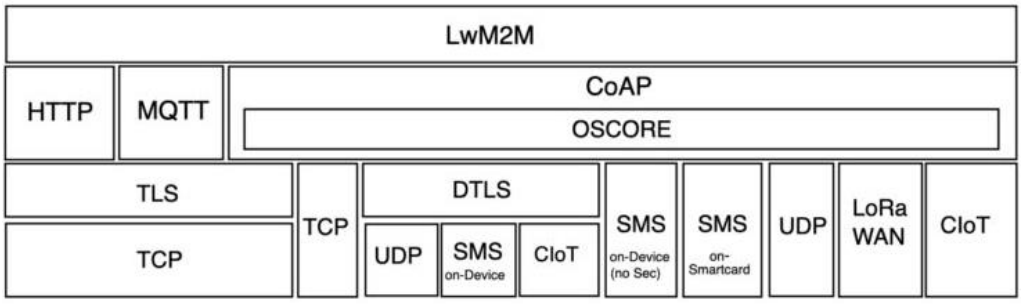

Figure 6.5 - LwM2M protocol stack

When OSCORE is used, the workflow described above can be further detailed as follows.

- The LwM2M Client and the Bootstrap Server has a pre-established OSCORE Security Context CTX1 (see Section 3.7.1). Hence, they use it to protect messages exchanged during the bootstrapping process.
- During the bootstrapping process, the Bootstrap Server provides the LwM2M Client with parameters to derive a second Security Context CTX2, to be used from then on when communicating with the LwM2M Server.
- The Bootstrap Server provides also the LwM2M Server with the same parameters for deriving CTX2. This provisioning typically happens on a dedicated, secure management channel, and further details are out of the scope of the standard.

- The LwM2M Client performs the registration process with the LwM2M Server. During the registration process, the two peers use CTX2 in order to protect exchanged messages with OSCORE.
- Following the registration process, the L2M2M Client and Server can engage in further message exchanges, again protected with OSCORE by using the Security Context CTX2.

This approach for establishing the Security Context CTX2 has the following limitations, due to the used key establishment approach as a secure provisioning performed by the Bootstrap Server.

- The keying material in the derived Security Context CTX2 does not enjoy forward secrecy. That is, it might generally happen that the long-term secrets that the LwM2M Client and Server rely on during the key establishment phase get compromised. For the LwM2M Client, such a secret is the OSCORE Master Secret in CTX1 used with the Bootstrap Server. For the LwM2M Server it can be, e.g., the private key of the LwM2M Server as used in a previous secure channel establishment with the Bootstrap Server. If such secrets get compromised, an adversary would be able to also derive CTX2 and thus get access to (past) communications protected with CTX2.

- The details about how the LwM2M Client and Server have to use OSCORE according to CTX2 are basically dictated by the Bootstrap Server, e.g., by following a pre-configured local policy. That is, the LwM2M Client and Server are the parties intended to use CTX2, but yet play no role in determining and negotiating which particular cryptographic algorithms to use when communicating with OSCORE.

With the goal of overcoming the limitations above, work has been ongoing to integrate the EDHOC key establishment protocol [SEL21] (see Section 6.3) within the LwM2M framework, in support to its OSCORE-based workflow. As discussed above, the original approach relies on the Bootstrap Server to provide both the LwM2M Client and Server with information to derive the OSCORE Security Context CTX2. Instead, the alternative approach would rely on the Bootstrap Server providing both the LwM2M Client and Server with information on how to run EDHOC, as a preliminary step before the registration process takes place.

This also requires defining a new LwM2M Object for providing such information to the LwM2M Client during the bootstrapping process. The follow-up EDHOC execution between LwM2M Client and Server allows them to mutually authenticate through their credentials, directly negotiate parameters affecting the later OSCORE-based communication, and most importantly derive the OSCORE Security Context CTX2 with forward secrecy. Note that the approach described in Section 6.3.1 can also be used, in order to attain a single CoAP message combining the last EDHOC message and the registration message, both from the LwM2M client. Also, one may want to use EDHOC from the start, i.e., between the LwM2M Client and Bootstrap Server to establish the Security Context CTX1, rather than having it pre-configured and without forward secrecy.

Finally, this approach can “scale up” with the Bootstrap Server providing the LwM2M Client with information to run EDHOC not only with the LwM2M Server, but also with an Application Server external to the LwM2M domain. That is, the LwM2M Client would still use the Security Context CTX2 to protect communications with the LwM2M Server, while a different Security Context CTX3 to protect communications with the Application Server, with both Security Contexts established by executing EDHOC with the respective peers.

While the LwM2M Client might in principle communicate with an external Application Server on any available network path, it is reasonable for realistic deployments to force this to occur through the LwM2M Server acting as a CoAP forward-proxy. This ensures that the LwM2M Server continues communicating with the LwM2M Client using OSCORE, and thus can identify the exact LwM2M Client before forwarding a message out of the LwM2M domain. However, this requires that an OSCORE-protected message from the LwM2M Client and targeting the Application Server is also further OSCORE-protected for the LwM2M server, in a multi-layer

fashion. While this kind of nested OSCORE protection is not admitted in the original OSCORE specification, the approach described in Section 4.5 is a candidate to enable this feature in this and other relevant use cases.

7 Conclusion

This document is the second deliverable from WP3 "Network and System Security", and has provided a preliminary description of the network & system security solutions designed and developed in the SIFIS-Home project. The presented content reflects the outcome of the WP3 activities carried out during the first half of the project, i.e., up until March 2022.

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. In particular, they have been explicitly related to the pertaining requirements documented in deliverable D1.2 "Final Architecture Requirements Report", as well as to the pertaining architecture components documented in deliverable D1.3 "Initial Component, Architecture, and Intercommunication Design". Where applicable, the description of a security solution pointed also to our open-source implementation, and to our related standardization proposals in the international body IETF.

During the second half of the SIFIS-Home project, we will progress the design and development of the security solutions from WP3 described in this document. Furthermore, a selection of such security solutions will be considered for integration in the demonstrators developed in WP5 "Integration, Testing and Demonstration", as well as in the project pilot developed in WP6 "Smart Home Pilot Use Case". Finally, we will continue the ongoing standardization work concerning the security solutions from WP3, with particular reference to the already targeted international body IETF.

A final description of the security solutions designed and developed in WP3 will be provided in deliverable D3.3 "Final report on Network and System Security Solutions". This will be released in June 2023, and it will update and obsolete the present document, thus acting as final comprehensive description of WP3 activities.

References

[AdaptiveDoS] Adaptive DoS Java implementation. Online: https://bitbucket.org/rhogl/adaptivedos

[AdaptiveDoSContiki] Adaptive DoS Contiki-NG implementation. Online: https://bitbucket.org/rhogl/contiki-ng

[ACE-DEV] ACE framework Java Implementation. Online: https://bitbucket.org/marco-tiloca-sics/ace-java/src/master/

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

[AMS21a] C. Amsüss, Z. Shelby, M. Koster, C. Bormann, and P. van der Stok, "CoRE Resource Directory", Internet-Draft draft-ietf-core-resource-directory-28 (work in progress), IETF Secretariat, March 2021.

[AMS21b] C. Amsüss and M. Tiloca, "Cacheable OSCORE", Internet-Draft draft-amsuess-core-cachable-oscore-03 (work in progress), IETF Secretariat, November 2021.

[BAL21] WSO2 Balana implementation: https://github.com/wso2/balana (fetched April 2021)

[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

[BOE08] S. Boeyen, S. Santesson, T. Polk, Russ Housley, S. Farrell and D. Cooper, "CBOR Encoded X.509 Certificates (C509 Certificates)", RFC 5280 (Proposed Standard), Internet Engineering Task Force, RFC Editor, May 2008.

[BOR13] C. Bormann and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049 (Proposed Standard), Internet Engineering Task Force, RFC Editor, October 2013.

[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/

[CAR16] E. Carniani, D. D'Arenzo, A. Lazouski, F. Martinelli, P. Mori, "Usage Control on Cloud Systems", Future Generation Computer Systems 63(C): Elsevier Science, pp. 37-55, 2016.

[Contiki-NG] The Contiki-NG Operating System. Online: https://www.contiki-ng.org/

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

[D1.3] SIFIS-Home Deliverable D1.3 "Initial Component, Architecture, and Intercommunication Design", September 2021.

[D3.1] SIFIS-Home Deliverable D3.1 "Analyses and Feedback on Architecture Requirements and Goals", May 2021.

[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/

[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-05 (work in progress), IETF Secretariat, November 2021.

[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.

[DU17] M. Du, F. Li, G. Zheng and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS ‘17), ACM, pp. 1285–1298, October 2017.

[EDHOC-DEV] EDHOC Java Implementation. Online: https://github.com/rikard-sics/californium/tree/edhoc

[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.

[GEH15] C. Gehrmann, M. Tiloca, and R. Höglund, "SMACK: Short message authentication check against battery exhaustion in the Internet of Things", in IEEE SECON 2015. IEEE, pp. 274-282, June 2015.

[GER21] 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)", Internet-Draft draft-ietf-ace-dtls-authorize-18 (work in progress), IETF Secretariat, June 2021.

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

[GOSC-DEVb] Implementation of Group OSCORE for Contiki-NG. Online: https://github.com/Gunzter/contiki-ng/

[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).

[GÜN21] F. Günther, M. Thomson and C.A. Wood, "Usage Limits on AEAD Algorithms", Internet Draft draft-irtf-cfrg-aead-limits-03 (work in progress), IETF Secretariat, December 2021.

[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.

[HÖG21] R. Höglund and M. Tiloca, "Key Update for OSCORE (KUDOS)", Internet-Draft draft-ietf-core-oscore-key-update-00 (work in progress), IETF Secretariat, December 2021.

[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.

[KOS19] M. Koster, A. Keränen and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application

Protocol (CoAP)", Internet-Draft draft-ietf-core-coap-pubsub-09 (work in progress), IETF Secretariat, September 2019.

[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://webee.technion.ac.il/~hugo/sigma-pdf.pdf

[KRA10] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869 (Informational), Internet Research Task Force, RFC Editor, May 2010.

[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.

[MAT22] 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-03 (work in progress), IETF Secretariat, January 2022.

[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

[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

[OSC-DEV] Implementation of OSCORE for Contiki-NG. Online: https://github.com/Gunzter/contiki-ng/

[PAL21a] F. Palombini, L. Seitz, G. Selander and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", Internet-Draft draft-ietf-ace-oscore-profile-19 (work in progress), IETF Secretariat, May 2021.

[PAL21b] F. Palombini and M. Tiloca, "Key Provisioning for Group Communication using ACE", Internet-Draft draft-ietf-ace-key-groupcomm-14 (work in progress), IETF Secretariat, October 2021.

[PAL21c] F. Palombini, M. Tiloca, R. Höglund, S. Hristozov and G. Selander, "Profiling EDHOC for CoAP and OSCORE", Internet-Draft draft-ietf-core-oscore-edhoc-02 (work in progress), IETF Secretariat, November 2021.

[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/

[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.

[SCH17] J. Schaad, "CBOR Object Signing and Encryption (COSE)", RFC 8152 (Proposed Standard), RFC Editor, July 2017.

[SEI13] L. Seitz, G. Selander, and C. Gehrmann, "Authorization framework for the Internet-of-Things" in D-SPAN workshop of IEEE WoWMoM 2013. IEEE Computer Society, pp. 1-6, June 2013.

[SEI21] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE- OAuth)", Internet Draft draft-ietf-ace-oauth-authz-46 (work in progress), IETF Secretariat, November 2021.

[SEL19] G. Selander, J. Mattsson, F. Palombini and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC8613 (Proposed Standard), Internet Engineering Task Force, RFC Editor, July 2019.

[SEL21] G. Selander, J. Mattsson and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Internet-Draft draft-ietf-lake-edhoc-12 (work in progress), IETF Secretariat, October 2021.

[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.

[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

[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.

[TIL18] M. Tiloca, R. Höglund, and S. Al Atiiq, "SARDOS: Self-Adaptive Reaction Against Denial of Service in the Internet of Things", in IEEE IoTSMS 2018. IEEE, June 2018

[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

[TIL21a] M. Tiloca, G. Selander, F. Palombini, J. Mattsson and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Internet-Draft draft-ietf-core-oscore-groupcomm-13 (work in progress), IETF Secretariat, October 2021.

[TIL21b] M. Tiloca, J. Park and F. Palombini, "Key Management for OSCORE Groups in ACE", Internet-Draft draft-ietf-ace-key-groupcomm-oscore-12 (work in progress), IETF Secretariat, October 2021.

[TIL21c] 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-04 (work in progress), IETF Secretariat, October 2021.

[TIL21d] 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-10 (work in progress), IETF Secretariat, October 2021.

[TIL21e] M. Tiloca, R. Höglund, C. Amsüss and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Internet-Draft draft-ietf-core-observe-multicast-notifications-02 (work in progress), IETF Secretariat, October 2021.

[TIL21f] M. Tiloca and E. Dijk, "Proxy Operations for CoAP Group Communication", Internet-Draft draft-tiloca-core-groupcomm-proxy-05 (work in progress), IETF Secretariat, October 2021.

[TIL21g] M. Tiloca and R. Höglund, "OSCORE-capable Proxies", Internet-Draft draft-tiloca-core-oscore-capable-proxies-01 (work in progress), IETF Secretariat, October 2021.

[TIL21h] 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-06 (work in progress), IETF Secretariat, July 2021.

[TIL21i] M. Tiloca, L. Seitz, 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-00 (work in progress), IETF Secretariat, November 2021.

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

[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.

Annex A: Glossary

Acronym	Definition
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
DoS	Denial of Service
DDoS	Distributed Denial of Service
DTLS	Datagram Transport Layer Security
EDHOC	Ephemeral Diffie-Hellman Over COSE
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)
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
SMACK	Short Message Authentication ChecK

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