CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
RISE AB
G. Selander
F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen

November 02, 2020

Group OSCORE - Secure Group Communication for CoAP
draft-ietf-core-oscore-groupcomm-10

Abstract

This document defines Group Object Security for Constrained RESTful Environments (Group OSCORE), providing end-to-end security of CoAP messages exchanged between members of a group, e.g. sent over IP multicast. In particular, the described approach defines how OSCORE is used in a group communication setting to provide source authentication for CoAP group requests, sent by a client to multiple servers, and for protection of the corresponding CoAP responses.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol specifically designed for constrained devices and networks [RFC7228]. Group communication for CoAP [I-D.ietf-core-groupcomm-bis] addresses use cases where deployed devices benefit from a group communication model, for example to reduce latencies, improve performance and reduce bandwidth utilization. Use cases include lighting control, integrated building control, software and firmware updates, parameter and configuration updates, commissioning of constrained networks, and emergency multicast (see Appendix B). This specification defines the security protocol for Group communication for CoAP [I-D.ietf-core-groupcomm-bis].

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and provides end-to-end encryption, integrity, replay protection and binding of response to request between a sender and a recipient, independent of transport also in the presence of intermediaries. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields in a COSE object, which replaces the authenticated and encrypted fields in the protected message.

This document defines Group OSCORE, providing the same end-to-end security properties as OSCORE in the case where CoAP requests have multiple recipients. In particular, the described approach defines how OSCORE is used in a group communication setting to provide source authentication for CoAP group requests, sent by a client to multiple servers, and for protection of the corresponding CoAP responses.

Just like OSCORE, Group OSCORE is independent of transport layer and works wherever CoAP does. Group communication for CoAP [I-D.ietf-core-groupcomm-bis] uses UDP/IP multicast as the underlying data transport.

As 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 [RFC6347][I-D.ietf-tls-dtls13], between one client and one proxy (and vice versa), or between one proxy and one server (and vice versa), in order to protect the routing information of packets from observers. Note that DTLS does not define how to secure messages sent over IP multicast.

Group OSCORE defines two modes of operation:

o In the group mode, Group OSCORE requests and responses are digitally signed with the private key of the sender and the signature is embedded in the protected CoAP message. The group mode supports all COSE algorithms as well as signature verification by intermediaries. This mode is defined in Section 8 and MUST be supported.

o In the pairwise mode, two group members exchange Group OSCORE requests and responses over unicast, and the messages are protected with symmetric keys. These symmetric keys are derived from Diffie-Hellman shared secrets, calculated with the asymmetric keys of the sender and recipient, allowing for shorter integrity tags and therefore lower message overhead. This mode is defined in Section 9 and is OPTIONAL to support.

Both modes provide source authentication of CoAP messages. The application decides what mode to use, potentially on a per-message basis. Such decisions can be based, for instance, on pre-configured policies or dynamic assessing of the target recipient and/or resource, among other things. One important case is when requests are protected with the group mode, and responses with the pairwise mode. Since such responses convey shorter integrity tags instead of bigger, full-fledged signatures, this significantly reduces the message overhead in case of many responses to one request.

A special deployment of Group OSCORE is to use pairwise mode only. For example, consider the case of a constrained-node network [RFC7228] with a large number of CoAP endpoints and the objective to establish secure communication between any pair of endpoints with a small provisioning effort and message overhead. Since the total number of security associations that needs to be established grows with the square of the number of nodes, it is desirable to restrict

the provisioned keying material. Moreover, a key establishment protocol would need to be executed for each security association. One solution to this is to deploy Group OSCORE, with the endpoints being part of a group, and use the pairwise mode. This solution assumes a trusted third party called Group Manager (see Section 3), but has the benefit of restricting the symmetric keying material while distributing only the public key of each group member. After that, a CoAP endpoint can locally derive the OSCORE Security Context for the other endpoint in the group, and protect CoAP communications with very low overhead [I-D.ietf-lwig-security-protocol-comparison].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252] including "endpoint", "client", "server", "sender" and "recipient"; group communication for CoAP [I-D.ietf-core-groupcomm-bis]; CBOR [I-D.ietf-cbor-7049bis]; COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and related counter signatures [I-D.ietf-cose-countersign].

Readers are also expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE, such as "Security Context" and "Master Secret", defined in [RFC8613].

Terminology for constrained environments, such as "constrained device" and "constrained-node network", is defined in [RFC7228].

This document refers also to the following terminology.

o Keying material: data that is necessary to establish and maintain secure communication among endpoints. This includes, for instance, keys and IVs [RFC4949].

o Group: a set of endpoints that share group keying material and security parameters (Common Context, see Section 2). Unless specified otherwise, the term group used in this specification refers thus to a "security group" (see Section 2.1 of [I-D.ietf-core-groupcomm-bis]), not to be confused with "CoAP group" or "application group".

o Group Manager: entity responsible for a group. Each endpoint in a group communicates securely with the respective Group Manager,

which is neither required to be an actual group member nor to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 3.2.

- o Silent server: member of a group that never sends protected responses in reply to requests. For CoAP group communications, requests are normally sent without necessarily expecting a response. A silent server may send unprotected responses, as error responses reporting an OSCORE error. Note that an endpoint can implement both a silent server and a client, i.e. the two roles are independent. An endpoint acting only as a silent server performs only Group OSCORE processing on incoming requests. Silent servers maintain less keying material and in particular do not have a Sender Context for the group. Since silent servers do not have a Sender ID, they cannot support the pairwise mode.

- o Group Identifier (Gid): identifier assigned to the group, unique within the set of groups of a given Group Manager.

- o Group request: CoAP request message sent by a client in the group to all servers in that group.

- o Source authentication: evidence that a received message in the group originated from a specific identified group member. This also provides assurance that the message was not tampered with by anyone, be it a different legitimate group member or an endpoint which is not a group member.

2. Security Context

This specification refers to a group as a set of endpoints sharing keying material and security parameters for executing the Group OSCORE protocol (see Section 1.1). Each endpoint which is member of a group maintains a Security Context as defined in Section 3 of [RFC8613], extended as follows (see Figure 1):

- o One Common Context, shared by all the endpoints in the group. Two new parameters are included in the Common Context, namely Counter Signature Algorithm and Counter Signature Parameters. These relate to the computation of counter signatures, when messages are protected using the group mode (see Section 8).

  If the pairwise mode is supported, the Common Context is further extended with two new parameters, namely Secret Derivation Algorithm and Secret Derivation Parameters. These relate to the derivation of a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.3.1) to protect messages with the pairwise mode (see Section 9).

- One Sender Context, extended with the endpoint's private key. The private key is used to sign the message in group mode, and for deriving the pairwise keys in pairwise mode (see Section 2.3). If the pairwise mode is supported, the Sender Context is also extended with the Pairwise Sender Keys associated to the other endpoints (see Section 2.3). 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. It is not necessary to maintain Recipient Contexts associated to endpoints from which messages are not (expected to be) received. The Recipient Context is extended with the public key of the associated endpoint, used to verify the signature in group mode and for deriving the pairwise keys in pairwise mode (see Section 2.3). If the pairwise mode is supported, then the Recipient Context is also extended with the Pairwise Recipient Key associated to the other endpoint (see Section 2.3).

Context Component	New Information Elements
Common Context	Counter Signature Algorithm Counter Signature Parameters *Secret Derivation Algorithm *Secret Derivation Parameters
Sender Context	Endpoint's own private key *Pairwise Sender Keys for the other endpoints
Each Recipient Context	Public key of the other endpoint *Pairwise Recipient Key of the other endpoint

Figure 1: Additions to the OSCORE Security Context. Optional additions are labeled with an asterisk.

Further details about the Security Context of Group OSCORE are provided in the remainder of this section. How the Security Context is established by the group members is out of scope for this specification, but if there is more than one Security Context applicable to a message, then the endpoints MUST be able to tell which Security Context was latest established.

The default setting for how to manage information about the group is described in terms of a Group Manager (see Section 3).

2.1. Common Context

The Common Context may be acquired from the Group Manager (see Section 3). The following sections define how the Common Context is extended, compared to [RFC8613].

2.1.1. ID Context

The ID Context parameter (see Sections 3.3 and 5.1 of [RFC8613]) in the Common Context SHALL contain the Group Identifier (Gid) of the group. The choice of the Gid format is application specific. An example of specific formatting of the Gid is given in Appendix C. The application needs to specify how to handle potential collisions between Gids (see Section 10.5).

2.1.2. Counter Signature Algorithm

Counter Signature Algorithm identifies the digital signature algorithm used to compute a counter signature on the COSE object (see Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign]), when messages are protected using the group mode (see Section 8).

This parameter is immutable once the Common Context is established. Counter Signature Algorithm MUST take value from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms]. The value is associated to a COSE key type, as specified in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms]. COSE capabilities for algorithms are defined in Section 8 of [I-D.ietf-cose-rfc8152bis-algs].

The EdDSA signature algorithm and the elliptic curve Ed25519 [RFC8032] are mandatory to implement. If elliptic curve signatures are used, it is RECOMMENDED to implement deterministic signatures with additional randomness as specified in [I-D.mattsson-cfrg-det-sigs-with-noise].

2.1.3. Counter Signature Parameters

Counter Signature Parameters identifies the parameters associated to the digital signature algorithm specified in Counter Signature Algorithm. This parameter is immutable once the Common Context is established.

This parameter is a CBOR array including the following two elements, whose exact structure and value depend on the value of Counter Signature Algorithm:

- The first element is the array of COSE capabilities for Counter Signature Algorithm, as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms] (see Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs]).

- The second element is the array of COSE capabilities for the COSE key type associated to Counter Signature Algorithm, as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types] (see Section 8.2 of [I-D.ietf-cose-rfc8152bis-algs]).

Examples of Counter Signature Parameters are in Appendix G.

2.1.4. Secret Derivation Algorithm

Secret Derivation Algorithm identifies the elliptic curve Diffie-Hellman algorithm used to derive a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.3.1) to protect messages with the pairwise mode (see Section 9).

This parameter is immutable once the Common Context is established. Secret Derivation Algorithm MUST take value from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms]. The value is associated to a COSE key type, as specified in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms]. COSE capabilities for algorithms are defined in Section 8 of [I-D.ietf-cose-rfc8152bis-algs].

For endpoints that support the pairwise mode, the ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs] and the X25519 curve [RFC7748] are mandatory to implement.

2.1.5. Secret Derivation Parameters

Secret Derivation Parameters identifies the parameters associated to the elliptic curve Diffie-Hellman algorithm specified in Secret Derivation Algorithm. This parameter is immutable once the Common Context is established.

This parameter is a CBOR array including the following two elements, whose exact structure and value depend on the value of Secret Derivation Algorithm:

- The first element is the array of COSE capabilities for Secret Derivation Algorithm, as specified for that algorithm in the

"Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms] (see Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs]).

- o The second element is the array of COSE capabilities for the COSE key type associated to Secret Derivation Algorithm, as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types] (see Section 8.2 of [I-D.ietf-cose-rfc8152bis-algs]).

Examples of Secret Derivation Parameters are in Appendix G.

2.2. Sender Context and Recipient Context

OSCORE specifies the derivation of Sender Context and Recipient Context, specifically of Sender/Recipient Keys and Common IV, from a set of input parameters (see Section 3.2 of [RFC8613]). This derivation applies also to Group OSCORE, and the mandatory-to-implement HKDF and AEAD algorithms are the same as in [RFC8613]. The Sender ID SHALL be unique for each endpoint in a group with a fixed Master Secret, Master Salt and Group Identifier (see Section 3.3 of [RFC8613]).

For Group OSCORE, the Sender Context and Recipient Context additionally contain asymmetric keys, as described previously in Section 2. The private/public key pair of the sender can, for example, be generated by the endpoint or provisioned during manufacturing.

With the exception of the public key of the sender endpoint, a receiver endpoint can derive a complete Security Context from a received Group OSCORE message and the Common Context. The public keys in the Recipient Contexts can be retrieved from the Group Manager (see Section 3) upon joining the group. A public key can alternatively be acquired from the Group Manager at a later time, for example the first time a message is received from a particular endpoint in the group (see Section 8.2 and Section 8.4).

For severely constrained devices, it may be not feasible to simultaneously handle the ongoing processing of a recently received message in parallel with the retrieval of the sender endpoint's public key. Such devices can be configured to drop a received message for which there is no (complete) Recipient Context, and retrieve the sender endpoint's public key in order to have it available to verify subsequent messages from that endpoint.

Furthermore, sufficiently large replay windows should be considered, to handle Partial IV values moving forward fast. This can happen

when a client engages in frequent or long sequences of one-to-one exchanges with servers in the group, such as a large number of block-wise transfers to a single server. When receiving following group requests from that client, other servers in the group may believe to have lost synchronization with the client's Sender Sequence Number. If these servers use an Echo exchange to re-gain synchronization (see Appendix E.3), this in itself may consume a considerable amount of client's Sender Sequence Numbers, hence later resulting in the servers possibly performing a new Echo exchange.

2.3. Pairwise Keys

Certain signature schemes, such as EdDSA and ECDSA, support a secure combined signature and encryption scheme. This section specifies the derivation of "pairwise keys", for use in the pairwise mode of Group OSCORE defined in Section 9.

2.3.1. Derivation of Pairwise Keys

Using the Group OSCORE Security Context (see Section 2), a group member can derive AEAD keys to protect point-to-point communication between itself and any other endpoint in the group. The same AEAD algorithm as in the group mode is used. The key derivation of these so-called pairwise keys follows the same construction as in Section 3.2.1 of [RFC8613]:

```
Pairwise Recipient Key = HKDF(Recipient Key, Shared Secret, info, L)
Pairwise Sender Key    = HKDF(Sender Key, Shared Secret, info, L)
```

where:

o The Pairwise Recipient Key is the AEAD key for processing incoming messages from endpoint X.

o The Pairwise Sender Key is the AEAD key for processing outgoing messages addressed to endpoint X.

o HKDF is the HKDF algorithm specified by Secret Derivation Algorithm from the Common Context (see Section 2.1.4).

o The Shared Secret is computed as a static-static Diffie-Hellman shared secret [NIST-800-56A], where the endpoint uses its private key and the public key of the other endpoint X.

o The Recipient Key and the public key are from the Recipient Context associated to endpoint X.

o The Sender Key and private key are from the Sender Context.

o info and L are defined as in Section 3.2.1 of [RFC8613].

If EdDSA asymmetric keys are used, the Edward coordinates are mapped to Montgomery coordinates using the maps defined in Sections 4.1 and 4.2 of [RFC7748], before using the X25519 and X448 functions defined in Section 5 of [RFC7748].

After establishing a partially or completely new Security Context (see Section 3.1 and Section 2.4), the old pairwise keys MUST be deleted. Since new Sender/Recipient Keys are derived from the new group keying material (see Section 2.2), every group member MUST use the new Sender/Recipient Keys when deriving new pairwise keys.

As long as any two group members preserve the same asymmetric keys, their Diffie-Hellman shared secret does not change across updates of the group keying material.

2.3.2. Usage of Sequence Numbers

When using any of its Pairwise Sender Keys, a sender endpoint including the 'Partial IV' parameter in the protected message MUST use the current fresh value of the Sender Sequence Number from its Sender Context (see Section 2.2). That is, the same Sender Sequence Number space is used for all outgoing messages protected with Group OSCORE, thus limiting both storage and complexity.

On the other hand, when combining group and pairwise communication modes, this may result in the Partial IV values moving forward more often. This can happen when a client engages in frequent or long sequences of one-to-one exchanges with servers in the group, by sending requests over unicast.

2.3.3. Security Context for Pairwise Mode

If the pairwise mode is supported, the Security Context additionally includes Secret Derivation Algorithm, Secret Derivation Parameters and the pairwise keys, as described at the beginning of Section 2.

The pairwise keys as well as the shared secrets used in their derivation (see Section 2.3.1) may be stored in memory or recomputed every time they are needed. The shared secret changes only when a public/private key pair used for its derivation changes, which results in the pairwise keys also changing. Additionally, the pairwise keys change if the Sender ID changes or if a new Security Context is established for the group (see Section 2.4.3). In order to optimize protocol performance, an endpoint may store the derived pairwise keys for easy retrieval.

In the pairwise mode, the Sender Context includes the Pairwise Sender Keys to use with the other endpoints (see Figure 1). In order to identify the right key to use, the Pairwise Sender Key for endpoint X may be associated to the Recipient ID of endpoint X, as defined in the Recipient Context (i.e. the Sender ID from the point of view of endpoint X). In this way, the Recipient ID can be used to lookup for the right Pairwise Sender Key. This association may be implemented in different ways, e.g. by storing the pair (Recipient ID, Pairwise Sender Key) or linking a Pairwise Sender Key to a Recipient Context.

2.4. Update of Security Context

It is RECOMMENDED that the immutable part of the Security Context is stored in non-volatile memory, or that it can otherwise be reliably accessed throughout the operation of the group, e.g. after a device reboots. However, also immutable parts of the Security Context may need to be updated, for example due to scheduled key renewal, new or re-joining members in the group, or the fact that the endpoint changes Sender ID (see Section 2.4.3).

On the other hand, the mutable parts of the Security Context are updated by the endpoint when executing the security protocol, but may nevertheless become outdated, e.g. due to loss of the mutable Security Context (see Section 2.4.1) or exhaustion of Sender Sequence Numbers (see Section 2.4.2).

If it is not feasible or practically possible to store and maintain up-to-date the mutable part in non-volatile memory (e.g., due to limited number of write operations), the endpoint MUST be able to detect a loss of the mutable Security Context.

When a loss of mutable Security Context is detected (e.g., following a reboot), the endpoint MUST NOT protect further messages using this Security Context to avoid reusing a nonce with the same AEAD key, and SHOULD instead retrieve new security parameters from the Group Manager (see Section 2.4.1).

2.4.1. Loss of Mutable Security Context

An endpoint that has lost its mutable Security Context, e.g. due to a reboot, needs to prevent the re-use of a nonce with the same AEAD key, and to handle incoming replayed messages.

To this end, after a loss of mutable Security Context, the endpoint SHOULD inform the Group Manager, retrieve new Security Context parameters from the Group Manager (see Section 2.4.3), and use them to derive a new Sender Context (see Section 2.2). In particular, regardless the exact actions taken by the Group Manager, the endpoint

resets its Sender Sequence Number to 0, and derives a new Sender Key. This is in turn used to possibly derive new Pairwise Sender Keys.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

If an endpoint is not able to establish an updated Sender Context, e.g. because of lack of connectivity with the Group Manager, the endpoint MUST NOT protect further messages using the current Security Context.

In order to handle the update of Replay Window in Recipient Contexts, three approaches are discussed in Appendix E. In particular, the approach specified in Appendix E.3 and based on the Echo Option [I-D.ietf-core-echo-request-tag] is a variant of the approach defined in Appendix B.1.2 of [RFC8613] as applicable to Group OSCORE.

2.4.2. Exhaustion of Sender Sequence Number

An endpoint can eventually exhaust the Sender Sequence Number, which is incremented for each new outgoing message including a Partial IV. This is the case for group requests, Observe notifications [RFC7641] and, optionally, any other response.

Implementations MUST be able to detect an exhaustion of Sender Sequence Number, after the endpoint has consumed the largest usable value. If an implementation's integers support wrapping addition, the implementation MUST treat Sender Sequence Number as exhausted when a wrap-around is detected.

Upon exhausting the Sender Sequence Numbers, the endpoint MUST NOT use this Security Context to protect further messages including a Partial IV.

The endpoint SHOULD inform the Group Manager, retrieve new Security Context parameters from the Group Manager (see Section 2.4.3), and use them to derive a new Sender Context (see Section 2.2). In particular, regardless the exact actions taken by the Group Manager, the endpoint resets its Sender Sequence Number to 0, and derives a new Sender Key. This is in turn used to possibly derive new Pairwise Sender Keys.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

2.4.3. Retrieving New Security Context Parameters

The Group Manager can assist an endpoint with an incomplete Sender Context to retrieve missing data of the Security Context and thereby become fully operational in the group again. The two main options for the Group Manager are described in this section: i) assignment of a new Sender ID to the endpoint (see Section 2.4.3.1); and ii) establishment of a new Security Context for the group (see Section 2.4.3.2). Update of Replay Window in Recipient Contexts is discussed in Section 6.1.

As group membership changes, or as group members get new Sender IDs (see Section 2.4.3.1) so do the relevant Recipient IDs that the other endpoints need to keep track of. As a consequence, group members may end up retaining stale Recipient Contexts, that are no longer useful to verify incoming secure messages.

The Recipient ID ('kid') SHOULD NOT be considered as a persistent and reliable indicator of a group member. Such an indication can be achieved only by using that member's public key, when verifying countersignatures of received messages (in group mode), or when verifying messages integrity-protected with pairwise keying material derived from asymmetric keys (in pairwise mode).

Furthermore, applications MAY define policies to: i) delete (long-)unused Recipient Contexts and reduce the impact on storage space; as well as ii) check with the Group Manager that a public key is currently the one associated to a 'kid' value, after a number of consecutive failed verifications.

2.4.3.1. New Sender ID for the Endpoint

The Group Manager may assign a new Sender ID to an endpoint, while leaving the Gid, Master Secret and Master Salt unchanged in the group. In this case, the Group Manager MUST assign a Sender ID that has never been assigned before in the group.

Having retrieved the new Sender ID, and potentially other missing data of the immutable Security Context, the endpoint can derive a new Sender Context (see Section 2.2). When doing so, the endpoint re-initilizes the Sender Sequence Number in its Sender Context to 0.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

The assignment of a new Sender ID may be the result of different processes. The endpoint may request a new Sender ID, e.g. because of exhaustion of Sender Sequence Numbers (see Section 2.4.2). An

endpoint may request to re-join the group, e.g. because of losing its mutable Security Context (see Section 2.4.1), and receive as response a new Sender ID together with the latest immutable Security Context.

For the other group members, the Recipient Context corresponding to the old Sender ID becomes stale (see Section 3.1).

2.4.3.2. New Security Context for the Group

The Group Manager may establish a new Security Context for the group (see Section 3.1). The Group Manager does not necessarily establish a new Security Context for the group if one member has an outdated Security Context (see Section 2.4.3.1), unless that was already planned or required for other reasons.

All the group members need to acquire new Security Context parameters from the Group Manager. Once having acquired new Security Context parameters, each group member performs the following actions.

- From then on, it MUST NOT use the current Security Context to start processing new messages for the considered group.

- It completes any ongoing message processing for the considered group.

- It derives and install a new Security Context. In particular:

  * It re-derives the keying material stored in its Sender Context and Recipient Contexts (see Section 2.2). The Master Salt used for the re-derivations is the updated Master Salt parameter if provided by the Group Manager, or the empty byte string otherwise.

  * It resets to 0 its Sender Sequence Number in its Sender Context.

  * It re-initializes the Replay Window of each Recipient Context.

  * It resets to 0 the sequence number of each ongoing observation where it is an observer client and that it wants to keep active.

From then on, it can resume processing new messages for the considered group. In particular:

- It MUST use its latest installed Sender Context to protect outgoing messages.

- o It SHOULD use its latest installed Recipient Contexts to process incoming messages, unless application policies admit to temporarily retain and use the old, recent, Security Context (see Section 10.4.1).

The distribution of a new Gid and Master Secret may result in temporarily misaligned Security Contexts among group members. In particular, this may result in a group member not being able to process messages received right after a new Gid and Master Secret have been distributed. A discussion on practical consequences and possible ways to address them, as well as on how to handle the old Security Context, is provided in Section 10.4.

3. The Group Manager

As with OSCORE, endpoints communicating with Group OSCORE need to establish the relevant Security Context. Group OSCORE endpoints need to acquire OSCORE input parameters, information about the group(s) and about other endpoints in the group(s). This specification is based on the existence of an entity called Group Manager which is responsible for the group, but does not mandate how the Group Manager interacts with the group members. The responsibilities of the Group Manager are compiled in Section 3.2.

It is RECOMMENDED to use a Group Manager as described in [I-D.ietf-ace-key-groupcomm-oscore], where the join process is based on the ACE framework for authentication and authorization in constrained environments [I-D.ietf-ace-oauth-authz].

The Group Manager assigns unique Group Identifiers (Gids) to different groups under its control, as well as unique Sender IDs (and thereby Recipient IDs) to the members of those groups. The Group Manager MUST NOT reassign a Sender ID within the same group, and MUST NOT reassign a Gid value to the same group. According to a hierarchical approach, the Gid value assigned to a group is associated to a dedicated space for the values of Sender ID and Recipient ID of the members of that group.

In addition, the Group Manager maintains records of the public keys of endpoints in a group, and provides information about the group and its members to other members and selected roles. Upon nodes' joining, the Group Manager collects such public keys and MUST verify proof-of-possession of the respective private key.

An endpoint acquires group data such as the Gid and OSCORE input parameters including its own Sender ID from the Group Manager, and provides information about its public key to the Group Manager, for example upon joining the group.

A group member can retrieve from the Group Manager the public key and other information associated to another group member, with which it can generate the corresponding Recipient Context. An application can configure a group member to asynchronously retrieve information about Recipient Contexts, e.g. by Observing [RFC7641] a resource at the Group Manager to get updates on the group membership.

The Group Manager MAY serve additional entities acting as signature checkers, e.g. intermediary gateways. These entities do not join a group as members, but can retrieve public keys of group members from the Group Manager, in order to verify counter signatures of group messages. A signature checker MUST be authorized for retrieving public keys of members in a specific group from the Group Manager. To this end, the same method mentioned above based on the ACE framework [I-D.ietf-ace-oauth-authz] can be used.

3.1. Management of Group Keying Material

In order to establish a new Security Context for a group, a new Group Identifier (Gid) for that group and a new value for the Master Secret parameter MUST be generated. When distributing the new Gid and Master Secret, the Group Manager MAY distribute also a new value for the Master Salt parameter, and SHOULD preserve the current value of the Sender ID of each group member.

The Group Manager MUST NOT reassign a Gid value to the same group. That is, each group can have a given Gid at most once during its lifetime. An example of Gid format supporting this operation is provided in Appendix C.

The Group Manager MUST NOT reassign a previously used Sender ID ('kid') with the same Gid, Master Secret and Master Salt. Even if Gid and Master Secret are renewed as described in this section, the Group Manager MUST NOT reassign an endpoint's Sender ID ('kid') within a same group (see Section 2.4.3.1).

If required by the application (see Appendix A.1), it is RECOMMENDED to adopt a group key management scheme, and securely distribute a new value for the Gid and for the Master Secret parameter of the group's Security Context, before a new joining endpoint is added to the group or after a currently present endpoint leaves the group. This is necessary to preserve backward security and forward security in the group, if the application requires it.

The specific approach used to distribute new group data is out of the scope of this document. However, it is RECOMMENDED that the Group Manager supports the distribution of the new Gid and Master Secret

parameter to the group according to the Group Rekeying Process described in [I-D.ietf-ace-key-groupcomm-oscore].

3.2. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

1. Creating and managing OSCORE groups. This includes the assignment of a Gid to every newly created group, as well as ensuring uniqueness of Gids within the set of its OSCORE groups.

2. Defining policies for authorizing the joining of its OSCORE groups.

3. Handling the join process to add new endpoints as group members.

4. Establishing the Common Context part of the Security Context, and providing it to authorized group members during the join process, together with the corresponding Sender Context.

5. Generating and managing Sender IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process, or to current group members upon request of renewal. This includes ensuring that each Sender ID is unique within each of the OSCORE groups, and that it is not reassigned within the same group.

6. Defining communication policies for each of its OSCORE groups, and signalling them to new endpoints during the join process.

7. Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).

8. Providing the management keying material that a new endpoint requires to participate in the rekeying process, consistently with the key management scheme used in the group joined by the new endpoint.

9. Updating the Gid of its OSCORE groups, upon renewing the respective Security Context. This includes ensuring that the same Gid value is not reassigned to the same group.

10. Acting as key repository, in order to handle the public keys of the members of its OSCORE groups, and providing such public keys to other members of the same group upon request. The actual storage of public keys may be entrusted to a separate secure storage device or service.

11. Validating that the format and parameters of public keys of group members are consistent with the countersignature algorithm and related parameters used in the respective OSCORE group.

The Group Manager described in [I-D.ietf-ace-key-groupcomm-oscore] provides these functionalities.

4. The COSE Object

Building on Section 5 of [RFC8613], this section defines how to use COSE [I-D.ietf-cose-rfc8152bis-struct] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Associated Data (AEAD) algorithm. Unless otherwise specified, the following modifications apply for both the group mode and the pairwise mode of Group OSCORE.

4.1. Counter Signature

For the group mode only, the 'unprotected' field MUST additionally include the following parameter:

o COSE_CounterSignature0: its value is set to the counter signature of the COSE object, computed by the sender as described in Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign], by using its private key and according to the Counter Signature Algorithm and Counter Signature Parameters in the Security Context.

  In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 4.3.2 of this specification, and the ciphertext of the COSE object as payload.

4.2. The 'kid' and 'kid context' parameters

The value of the 'kid' parameter in the 'unprotected' field of response messages MUST be set to the Sender ID of the endpoint transmitting the message. That is, unlike in [RFC8613], the 'kid' parameter is always present in all messages, both requests and responses.

The value of the 'kid context' parameter in the 'unprotected' field of requests messages MUST be set to the ID Context, i.e. the Group Identifier value (Gid) of the group. That is, unlike in [RFC8613], the 'kid context' parameter is always present in requests.

4.3. external_aad

The external_aad of the Additional Authenticated Data (AAD) is different compared to OSCORE. In particular, there is one external_aad used for encryption (both in group mode and pairwise mode), and another external_aad used for signing (only in group mode).

4.3.1. external_aad for Encryption

The external_aad for encryption (see Section 4.3 of [I-D.ietf-cose-rfc8152bis-struct]), used both in group mode and pairwise mode, includes also the counter signature algorithm and related signature parameters, as well as the value of the 'kid context' in the COSE object of the request (see Figure 2).

```
external_aad = bstr .cbor aad_array

aad_array = [
   oscore_version : uint,
   algorithms : [alg_aead : int / tstr,
                 alg_countersign : int / tstr,
                 par_countersign : [countersign_alg_capab,
                                    countersign_key_type_capab],
                 par_countersign_key : countersign_key_type_capab],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr,
   request_kid_context : bstr
]
```

Figure 2: external_aad for Encryption

Compared with Section 5.4 of [RFC8613], the aad_array has the following differences.

- o The 'algorithms' array in the aad_array additionally includes:

  * 'alg_countersign', which specifies Counter Signature Algorithm from the Common Context (see Section 2.1.2). This parameter MUST encode the value of Counter Signature Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this counter signature algorithm.

  * 'par_countersign', which specifies the CBOR array Counter Signature Parameters from the Common Context (see Section 2.1.3). In particular:

      + 'countersign_alg_capab' is the array of COSE capabilities for the countersignature algorithm indicated in 'alg_countersign'. This is the first element of the CBOR array Counter Signature Parameters from the Common Context.

      + 'countersign_key_type_capab' is the array of COSE capabilities for the COSE key type used by the countersignature algorithm indicated in 'alg_countersign'. This is the second element of the CBOR array Counter Signature Parameters from the Common Context.

   * 'par_countersign_key', which specifies the parameters associated to the keys used with the countersignature algorithm indicated in 'alg_countersign'. These parameters are encoded as a CBOR array 'countersign_key_type_capab', whose exact structure and value depend on the value of 'alg_countersign'.

     In particular, 'countersign_key_type_capab' is the array of COSE capabilities for the COSE key type of the keys used with the countersignature algorithm. This is the second element of the CBOR array Counter Signature Parameters from the Common Context.

     Examples of 'par_countersign_key' are in Appendix G.

- The new element 'request_kid_context' contains the value of the 'kid context' in the COSE object of the request (see Section 4.2).

4.3.2. external_aad for Signing

The external_aad for signing (see Section 4.3 of [I-D.ietf-cose-rfc8152bis-struct]) used in group mode is identical to the external_aad for encryption (see Section 4.3.1) with the addition of the OSCORE option (see Figure 3).

```
external_aad = bstr .cbor aad_array

aad_array = [
   oscore_version : uint,
   algorithms : [alg_aead : int / tstr,
                 alg_countersign : int / tstr,
                 par_countersign : [countersign_alg_capab,
                                    countersign_key_type_capab],
                 par_countersign_key : countersign_key_type_capab],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr,
   request_kid_context : bstr,
   OSCORE_option: bstr
]
```

Figure 3: external_aad for Signing

Compared with Section 5.4 of [RFC8613] the aad_array additionally includes:

- the 'algorithms' array, as defined in the external_aad for encryption (see Section 4.3.1);

- the 'request_kid_context' element, as defined in the external_aad for encryption (see Section 4.3.1);

- the value of the OSCORE Option present in the protected message, encoded as a binary string.

Note for implementation: this construction requires the OSCORE option of the message to be generated before calculating the signature. Also, the aad_array needs to be large enough to contain the largest possible OSCORE option.

5. OSCORE Header Compression

The OSCORE header compression defined in Section 6 of [RFC8613] is used, with the following differences.

- The payload of the OSCORE message SHALL encode the ciphertext of the COSE_Encrypt0 object. In the group mode, the ciphertext above is concatenated with the value of the COSE_CounterSignature0 of the COSE object, computed as described in Section 4.1.

- This specification defines the usage of the sixth least significant bit, called the "Group Flag", in the first byte of the

OSCORE option containing the OSCORE flag bits. This flag bit is specified in Section 11.1.

- The Group Flag MUST be set to 1 if the OSCORE message is protected using the group mode (see Section 8).

- The Group Flag MUST be set to 0 if the OSCORE message is protected using the pairwise mode (see Section 9). The Group Flag MUST also be set to 0 for ordinary OSCORE messages processed according to [RFC8613].

5.1. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples of Group OSCORE used in group mode (see Section 5.1.1) or in pairwise mode (see Section 5.1.2).

The examples assume that the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the examples do not include the full CoAP unprotected message or the full Security Context, but only the input necessary to the compression mechanism, i.e. the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 5 and divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

The examples assume that the plaintext (see Section 5.3 of [RFC8613]) is 6 bytes long, and that the AEAD tag is 8 bytes long, hence resulting in a ciphertext which is 14 bytes long. When using the group mode, COUNTERSIGN denotes the COSE_CounterSignature0 byte string as described in Section 4, and is 64 bytes long.

5.1.1. Examples in Group Mode

- Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c

Before compression (96 bytes):

```
[
h'',
{ 4:h'25', 6:h'05', 10:h'44616c', 11:COUNTERSIGN },
h'aea0155667924dff8a24e4cb35b9'
]
```

```
   After compression (85 bytes):

   Flag byte: 0b00111001 = 0x39

   Option Value: 39 05 03 44 61 6c 25 (7 bytes)

   Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 COUNTERSIGN
   (14 bytes + size of COUNTERSIGN)
```

   o Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid =
      0x52 and no Partial IV.

```
   Before compression (88 bytes):

   [
   h'',
   { 4:h'52', 11:COUNTERSIGN },
   h'60b035059d9ef5667c5a0710823b'
   ]

   After compression (80 bytes):

   Flag byte: 0b00101000 = 0x28

   Option Value: 28 52 (2 bytes)

   Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b COUNTERSIGN
   (14 bytes + size of COUNTERSIGN)
```

5.1.2. Examples in Pairwise Mode

   o Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid =
      0x25, Partial IV = 5 and kid context = 0x44616c

```
   Before compression (32 bytes):

   [
   h'',
   { 4:h'25', 6:h'05', 10:h'44616c' },
   h'aea0155667924dff8a24e4cb35b9'
   ]
```

```
After compression (21 bytes):

Flag byte: 0b00011001 = 0x19

Option Value: 19 05 03 44 61 6c 25 (7 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)
```

o Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

```
Before compression (24 bytes):

[
h'',
{ 4:h'52'},
h'60b035059d9ef5667c5a0710823b'
]

After compression (16 bytes):

Flag byte: 0b00001000 = 0x08

Option Value: 08 52 (2 bytes)

Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b (14 bytes)
```

6. Message Binding, Sequence Numbers, Freshness and Replay Protection

The requirements and properties described in Section 7 of [RFC8613] also apply to OSCORE used in group communication. In particular, Group OSCORE provides message binding of responses to requests, which enables absolute freshness of responses that are not notifications, relative freshness of requests and notification responses, and replay protection of requests.

6.1. Update of Replay Window

A new server joining a group may not be aware of the current Partial IVs (Sender Sequence Numbers of the clients). Hence, when receiving a request from a particular client for the first time, the new server is not able to verify if that request is a replay. The same holds when a server loses its mutable Security Context (see Section 2.4.1), for instance after a device reboot.

The exact way to address this issue is application specific, and depends on the particular use case and its replay requirements. The

list of methods to handle the update of a Replay Window is part of the group communication policy, and different servers can use different methods. Appendix E describes three possible approaches that can be considered to address the issue discussed above.

Furthermore, when the Group Manager establishes a new Security Context for the group (see Section 2.4.3.2), every server re-initializes the Replay Window in each of its Recipient Contexts.

7. Message Reception

Upon receiving a protected message, a recipient endpoint retrieves a Security Context as in [RFC8613]. An endpoint MUST be able to distinguish between a Security Context to process OSCORE messages as in [RFC8613] and a Security Context to process Group OSCORE messages as defined in this specification.

To this end, an endpoint can take into account the different structure of the Security Context defined in Section 2, for example based on the presence of Counter Signature Algorithm in the Common Context. Alternatively implementations can use an additional parameter in the Security Context, to explicitly signal that it is intended for processing Group OSCORE messages.

If either of the following two conditions holds, a recipient endpoint MUST discard the incoming protected message:

- o The Group Flag is set to 1, and the recipient endpoint can not retrieve a Security Context which is both valid to process the message and also associated to an OSCORE group.

- o The Group Flag is set to 0, and the recipient endpoint retrieves a Security Context which is both valid to process the message and also associated to an OSCORE group, but the endpoint does not support the pairwise mode.

Otherwise, if a Security Context associated to an OSCORE group and valid to process the message is retrieved, the recipient endpoint processes the message with Group OSCORE, using the group mode (see Section 8) if the Group Flag is set to 1, or the pairwise mode (see Section 9) if the Group Flag is set to 0.

Note that, if the Group Flag is set to 0, and the recipient endpoint retrieves a Security Context which is valid to process the message but is not associated to an OSCORE group, then the message is processed according to [RFC8613].

8. Message Processing in Group Mode

When using the group mode, messages are protected and processed as specified in [RFC8613], with the modifications described in this section. The security objectives of the group mode are discussed in Appendix A.2. The group mode MUST be supported.

During all the steps of the message processing, an endpoint MUST use the same Security Context for the considered group. That is, an endpoint MUST NOT install a new Security Context for that group (see Section 2.4.3.2) until the message processing is completed.

The group mode MUST be used to protect group requests intended for multiple recipients or for the whole group. This includes both requests directly addressed to multiple recipients, e.g. sent by the client over multicast, as well as requests sent by the client over unicast to a proxy, that forwards them to the intended recipients over multicast [I-D.ietf-core-groupcomm-bis].

As per [RFC7252][I-D.ietf-core-groupcomm-bis], group requests sent over multicast MUST be Non-Confirmable, and thus are not retransmitted by the CoAP messaging layer. Instead, applications should store such outgoing messages for a pre-defined, sufficient amount of time, in order to correctly perform possible retransmissions at the application layer. According to Section 5.2.3 of [RFC7252], responses to Non-Confirmable group requests SHOULD also be Non-Confirmable, but endpoints MUST be prepared to receive Confirmable responses in reply to a Non-Confirmable group request. Confirmable group requests are acknowledged in non-multicast environments, as specified in [RFC7252].

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [RFC8613]. However, a recipient MUST stop processing and silently reject any message which is malformed and does not follow the format specified in Section 4, or which is not cryptographically validated in a successful way. In either case, it is RECOMMENDED that the recipient does not send back any error message. This prevents servers from replying with multiple error messages to a client sending a group request, so avoiding the risk of flooding and possibly congesting the network.

8.1. Protecting the Request

A client transmits a secure group request as described in Section 8.1 of [RFC8613], with the following modifications.

- In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.

- In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1.

- In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4 and Section 5 of this document. In particular, the payload of the OSCORE message includes also the counter signature.

8.1.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- If the client intends to keep the observation active beyond a possible change of Sender ID, the client MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. Even in case the client is individually rekeyed and receives a new Sender ID from the Group Manager (see Section 2.4.3.1), the client MUST NOT update the stored value associated to a particular Observe request.

- If the client intends to keep the observation active beyond a possible change of ID Context following a group rekeying (see Section 3.1), then the following applies.

  * The client MUST store the value of the 'kid context' parameter from the original Observe request, and retain it for the whole duration of the observation. Upon establishing a new Security Context with a new ID Context as Gid (see Section 2.4.3.2), the client MUST NOT update the stored value associated to a particular Observe request.

  * The client MUST store an invariant identifier of the group, which is immutable even in case the Security Context of the group is re-established. For example, this invariant identifier can be the "group name" in [I-D.ietf-ace-key-groupcomm-oscore], where it is used for joining the group and retrieving the current group keying material from the Group Manager.

    After a group rekeying, such an invariant information makes it simpler for the observer client to retrieve the current group

keying material from the Group Manager, in case the client has missed both the rekeying messages and the first observe notification protected with the new Security Context (see Section 8.3.1).

8.2. Verifying the Request

Upon receiving a secure group request with the Group Flag set to 1, following the procedure in Section 7, a server proceeds as described in Section 8.2 of [RFC8613], with the following modifications.

o In step 2, the decoding of the compressed COSE object follows Section 5 of this document. In particular:

  * If the server discards the request due to not retrieving a Security Context associated to the OSCORE group, the server MAY respond with a 4.02 (Bad Option) error. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.

  * If the received 'kid context' matches an existing ID Context (Gid) but the received 'kid' does not match any Recipient ID in this Security Context, then the server MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the associated public key. Such a configuration is application specific. If the application does not specify dynamic derivation of new Recipient Contexts, then the server SHALL stop processing the request.

o In step 4, the Additional Authenticated Data is modified as described in Section 4 of this document.

o In step 6, the server also verifies the counter signature using the public key of the client from the associated Recipient Context. In particular:

  * If the server does not have the public key of the client yet, the server MUST stop processing the request and MAY respond with a 5.03 (Service Unavailable) response. The response MAY include a Max-Age Option, indicating to the client the number of seconds after which to retry. If the Max-Age Option is not present, a retry time of 60 seconds will be assumed by the client, as default value defined in Section 5.10.5 of [RFC7252].

  * If the signature verification fails, the server SHALL stop processing the request and MAY respond with a 4.00 (Bad

Request) response. If the verification fails, the same steps are taken as if the decryption had failed. In particular, the Replay Window is only updated if both the signature verification and the decryption succeed.

- Additionally, if the used Recipient Context was created upon receiving this group request and the message is not verified successfully, the server MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the server's storage.

A server SHOULD NOT process a request if the received Recipient ID ('kid') is equal to its own Sender ID in its own Sender Context. For an example where this is not fulfilled, see Section 6.2.1 in [I-D.tiloca-core-observe-multicast-notifications].

8.2.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- The server MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. The server MUST NOT update the stored value of a 'kid' parameter associated to a particular Observe request, even in case the observer client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.4.3.1).

- The server MUST store the value of the 'kid context' parameter from the original Observe request, and retain it for the whole duration of the observation, beyond a possible change of ID Context following a group rekeying (see Section 3.1). That is, upon establishing a new Security Context with a new ID Context as Gid (see Section 2.4.3.2), the server MUST NOT update the stored value associated to the ongoing observation.

8.3. Protecting the Response

If a server generates a CoAP message in response to a Group OSCORE request, then the server SHALL follow the description in Section 8.3 of [RFC8613], with the modifications described in this section.

Note that the server always protects a response with the Sender Context from its latest Security Context, and that establishing a new Security Context resets the Sender Sequence Number to 0 (see Section 3.1).

- In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.

- In step 3, if the server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.1), 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. This prevents the AEAD nonce from the request from being reused.

- In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1. If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request (see Section 3.1), then the new ID Context MUST be included in the 'kid context' parameter of the response.

- In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 5 of this document. In particular, the payload of the OSCORE message includes also the counter signature.

8.3.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when protecting notifications for an ongoing observation.

- The server MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).

- The server MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid_context' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).

Furthermore, the server may have ongoing observations started by Observe requests protected with an old Security Context. After completing the establishment of a new Security Context, the server MUST protect the following notifications with the Sender Context of the new Security Context.

For each ongoing observation, the server MUST include in the first notification protected with the new Security Context also the 'kid context' parameter, which is set to the ID Context (Gid) of the new

Security Context. It is OPTIONAL for the server to include the ID Context (Gid) in the 'kid context' parameter also in further following notifications for those observations.

8.4. Verifying the Response

Upon receiving a secure response message with the Group Flag set to 1, following the procedure in Section 7, the client proceeds as described in Section 8.4 of [RFC8613], with the following modifications.

Note that a client may receive a response protected with a Security Context different from the one used to protect the corresponding group request, and that, upon the establishment of a new Security Context, the client re-initializes its replay windows in its Recipient Contexts (see Section 3.1).

- o In step 2, the decoding of the compressed COSE object is modified as described in Section 5 of this document. If the received 'kid context' matches an existing ID Context (Gid) but the received 'kid' does not match any Recipient ID in this Security Context, then the client MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the associated public key. If the application does not specify dynamic derivation of new Recipient Contexts, then the client SHALL stop processing the response.

- o In step 3, the Additional Authenticated Data is modified as described in Section 4 of this document.

- o In step 5, the client also verifies the counter signature using the public key of the server from the associated Recipient Context. If the verification fails, the same steps are taken as if the decryption had failed.

- o Additionally, if the used Recipient Context was created upon receiving this response and the message is not verified successfully, the client MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the client's storage.

8.4.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when verifying notifications for an ongoing observation.

- o The client MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.1.1), as value for the

'request_kid' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).

- The client MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.1.1), as value for the 'request_kid_context' parameter in the two external_aad structures (see Section 4.3.1 and Section 4.3.2).

This ensures that the client can correctly verify notifications, even in case it is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.4.3.1), as well as when it establishes a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 3.1).

9. Message Processing in Pairwise Mode

When using the pairwise mode of Group OSCORE, messages are protected and processed as in Section 8, with the modifications described in this section. The security objectives of the pairwise mode are discussed in Appendix A.2.

The pairwise mode takes advantage of an existing Security Context for the group mode to establish a Security Context shared exclusively with any other member. In order to use the pairwise mode, the signature scheme of the group mode MUST support a combined signature and encryption scheme. This can be, for example, signature using ECDSA, and encryption using AES-CCM with a key derived with ECDH.

The pairwise mode does not support the use of additional entities acting as verifiers of source authentication and integrity of group messages, such as intermediary gateways (see Section 3).

The pairwise mode MAY be supported. An endpoint implementing only a silent server does not support the pairwise mode.

If the signature algorithm used in the group supports ECDH (e.g., ECDSA, EdDSA), the pairwise mode MUST be supported by endpoints that use the CoAP Echo Option [I-D.ietf-core-echo-request-tag] and/or block-wise transfers [RFC7959], for instance for responses after the first block-wise request, which possibly targets all servers in the group and includes the CoAP Block2 option (see Section 2.3.6 of [I-D.ietf-core-groupcomm-bis]). This prevents the attack described in Section 10.7, which leverages requests sent over unicast to a single group member and protected with the group mode.

The pairwise mode protects messages between two members of a group, essentially following [RFC8613], but with the following notable differences:

- The 'kid' and 'kid context' parameters of the COSE object are used as defined in Section 4.2 of this document.

- The external_aad defined in Section 4.3.1 of this document is used for the encryption process.

- The Pairwise Sender/Recipient Keys used as Sender/Recipient keys are derived as defined in Section 2.3 of this document.

Senders MUST NOT use the pairwise mode to protect a message intended for multiple recipients. The pairwise mode is defined only between two endpoints and the keying material is thus only available to one recipient.

The Group Manager MAY indicate that the group uses also the pairwise mode, as part of the group data provided to candidate group members when joining the group.

9.1. Pre-Conditions

In order to protect an outgoing message in pairwise mode, the sender needs to know the public key and the Recipient ID for the recipient endpoint, as stored in the Recipient Context associated to that endpoint (see Pairwise Sender Context of Section 2.3.3).

Furthermore, the sender needs to know the individual address of the recipient endpoint. This information may not be known at any given point in time. For instance, right after having joined the group, a client may know the public key and Recipient ID for a given server, but not the addressing information required to reach it with an individual, one-to-one request.

To make addressing information of individual endpoints available, servers in the group MAY expose a resource to which a client can send a group request targeting a server or a set of servers, identified by their 'kid' value(s). The specified set may be empty, hence identifying all the servers in the group. Further details of such an interface are out of scope for this document.

9.2. Protecting the Request

When using the pairwise mode, the request is protected as defined in Section 8.1, with the following differences.

- The Group Flag MUST be set to 0.

- The used Sender Key is the Pairwise Sender Key (see Section 2.3).

- o The counter signature is not computed and therefore not included in the message. The payload of the protected request thus terminates with the encoded ciphertext of the COSE object, just like in [RFC8613].

Note that, like in the group mode, the external_aad for encryption is generated as in Section 4.3.1, and the Partial IV is the current fresh value of the client's Sender Sequence Number (see Section 2.3.2).

9.3. Verifying the Request

Upon receiving a request with the Group Flag set to 0, following the procedure in Section 7, the server MUST process it as defined in Section 8.2, with the following differences.

- o If the server discards the request due to not retrieving a Security Context associated to the OSCORE group or to not supporting the pairwise mode, the server MAY respond with a 4.02 (Bad Option) error. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.

- o If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.3.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.

- o The used Recipient Key is the Pairwise Recipient Key (see Section 2.3).

- o No verification of counter signature occurs, as there is none included in the message.

9.4. Protecting the Response

When using the pairwise mode, a response is protected as defined in Section 8.3, with the following differences.

- o The Group Flag MUST be set to 0.

- o The used Sender Key is the Pairwise Sender Key (see Section 2.3).

- o The counter signature is not computed and therefore not included in the message. The payload of the protected response thus terminates with the encoded ciphertext of the COSE object, just like in [RFC8613].

9.5. Verifying the Response

Upon receiving a response with the Group Flag set to 0, following the procedure in Section 7, the client MUST process it as defined in Section 8.4, with the following differences.

- o If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.3.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.

- o The used Recipient Key is the Pairwise Recipient Key (see Section 2.3).

- o No verification of counter signature occurs, as there is none included in the message.

10. Security Considerations

The same threat model discussed for OSCORE in Appendix D.1 of [RFC8613] holds for Group OSCORE. In addition, when using the group mode, source authentication of messages is explicitly ensured by means of counter signatures, as discussed in Section 10.1.

The same considerations on supporting Proxy operations discussed for OSCORE in Appendix D.2 of [RFC8613] hold for Group OSCORE.

The same considerations on protected message fields for OSCORE discussed in Appendix D.3 of [RFC8613] hold for Group OSCORE.

The same considerations on uniqueness of (key, nonce) pairs for OSCORE discussed in Appendix D.4 of [RFC8613] hold for Group OSCORE. This is further discussed in Section 10.2 of this document.

The same considerations on unprotected message fields for OSCORE discussed in Appendix D.5 of [RFC8613] hold for Group OSCORE, with the following difference. The counter signature included in a Group OSCORE message protected in group mode is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process. This is further discussed in Section 10.6 of this document.

As discussed in Section 6.2.3 of [I-D.ietf-core-groupcomm-bis], Group OSCORE addresses security attacks against CoAP listed in Sections 11.2-11.6 of [RFC7252], especially when run over IP multicast.

The rest of this section first discusses security aspects to be taken into account when using Group OSCORE. Then it goes through aspects covered in the security considerations of OSCORE (see Section 12 of [RFC8613]), and discusses how they hold when Group OSCORE is used.

10.1. Group-level Security

The group mode described in Section 8 relies on commonly shared group keying material to protect communication within a group. This has the following implications.

- o Messages are encrypted at a group level (group-level data confidentiality), i.e. they can be decrypted by any member of the group, but not by an external adversary or other external entities.

- o The AEAD algorithm provides only group authentication, i.e. it ensures that a message sent to a group has been sent by a member of that group, but not necessarily by the alleged sender. This is why source authentication of messages sent to a group is ensured through a counter signature, which is computed by the sender using its own private key and then appended to the message payload.

Instead, the pairwise mode described in Section 9 protects messages by using pairwise symmetric keys, derived from the static-static Diffie-Hellman shared secret computed from the asymmetric keys of the sender and recipient endpoint (see Section 2.3). Therefore, in the parwise mode, the AEAD algorithm provides both pairwise data-confidentiality and source authentication of messages, without using counter signatures.

The long-term storing of the Diffie-Hellman shared secret is a potential security issue. In fact, if the shared secret of two group members is leaked, a third group member can exploit it to impersonate any of those two group members, by deriving and using their pairwise key. The possibility of such leakage should be contemplated, as more likely to happen than the leakage of a private key, which could be rather protected at a significantly higher level than generic memory, e.g. by using a Trusted Platform Module. Therefore, there is a trade-off between the maximum amount of time a same shared secret is stored and the frequency of its re-computing.

Note that, even if an endpoint is authorized to be a group member and to take part in group communications, there is a risk that it behaves inappropriately. For instance, it can forward the content of messages in the group to unauthorized entities. However, in many use cases, the devices in the group belong to a common authority and are configured by a commissioner (see Appendix B), which results in a

practically limited risk and enables a prompt detection/reaction in case of misbehaving.

10.2. Uniqueness of (key, nonce)

The proof for uniqueness of (key, nonce) pairs in Appendix D.4 of [RFC8613] is also valid in group communication scenarios. That is, given an OSCORE group:

- o Uniqueness of Sender IDs within the group is enforced by the Group Manager, which never reassigns the same Sender ID within the same group.

- o The case A in Appendix D.4 of [RFC8613] concerns all group requests and responses including a Partial IV (e.g. Observe notifications). In this case, same considerations from [RFC8613] apply here as well.

- o The case B in Appendix D.4 of [RFC8613] concerns responses not including a Partial IV (e.g. single response to a group request). In this case, same considerations from [RFC8613] apply here as well.

As a consequence, each message encrypted/decrypted with the same Sender Key is processed by using a different (ID_PIV, PIV) pair. This means that nonces used by any fixed encrypting endpoint are unique. Thus, each message is processed with a different (key, nonce) pair.

10.3. Management of Group Keying Material

The approach described in this specification should take into account the risk of compromise of group members. In particular, this document specifies that a key management scheme for secure revocation and renewal of Security Contexts and group keying material should be adopted.

[I-D.ietf-ace-key-groupcomm-oscore] provides a simple rekeying scheme for renewing the Security Context in a group.

Alternative rekeying schemes which are more scalable with the group size may be needed in dynamic, large-scale groups where endpoints can join and leave at any time, in order to limit the impact on performance due to the Security Context and keying material update.

10.4. Update of Security Context and Key Rotation

A group member can receive a message shortly after the group has been rekeyed, and new security parameters and keying material have been distributed by the Group Manager.

This may result in a client using an old Security Context to protect a group request, and a server using a different new Security Context to protect a corresponding response. As a consequence, clients may receive a response protected with a Security Context different from the one used to protect the corresponding group request.

In particular, a server may first get a group request protected with the old Security Context, then install the new Security Context, and only after that produce a response to send back to the client. In such a case, as specified in Section 8.3, the server MUST protect the potential response using the new Security Context. Specifically, 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. This prevents the AEAD nonce from the request from being reused with the new Security Context.

The client will process that response using the new Security Context, provided that it has installed the new security parameters and keying material before the message processing.

In case block-wise transfer [RFC7959] is used, the same considerations from Section 7.2 of [I-D.ietf-ace-key-groupcomm] hold.

Furthermore, as described below, a group rekeying may temporarily result in misaligned Security Contexts between the sender and recipient of a same message.

10.4.1. Late Update on the Sender

In this case, the sender protects a message using the old Security Context, i.e. before having installed the new Security Context. However, the recipient receives the message after having installed the new Security Context, and is thus unable to correctly process it.

A possible way to ameliorate this issue is to preserve the old, recent, Security Context for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained Security Context as second attempt. This makes particular sense when the recipient is a client, that would hence be able to process incoming responses protected with the old, recent, Security Context used to protect the associated group request. Instead, a recipient server would better and more

simply discard an incoming group request which is not successfully processed with the new Security Context.

This tolerance preserves the processing of secure messages throughout a long-lasting key rotation, as group rekeying processes may likely take a long time to complete, especially in large scale groups. On the other hand, a former (compromised) group member can abusively take advantage of this, and send messages protected with the old retained Security Context. Therefore, a conservative application policy should not admit the retention of old Security Contexts.

10.4.2. Late Update on the Recipient

In this case, the sender protects a message using the new Security Context, but the recipient receives that message before having installed the new Security Context. Therefore, the recipient would not be able to correctly process the message and hence discards it.

If the recipient installs the new Security Context shortly after that and the sender endpoint retransmits the message, the former will still be able to receive and correctly process the message.

In any case, the recipient should actively ask the Group Manager for an updated Security Context according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

10.5. Collision of Group Identifiers

In case endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible for Group Identifiers of different groups to coincide.

This does not impair the security of the AEAD algorithm. In fact, as long as the Master Secret is different for different groups and this condition holds over time, AEAD keys are different among different groups.

The entity assigning an IP multicast address may help limiting the chances to experience such collisions of Group Identifiers. In particular, it may allow the Group Managers of groups using the same IP multicast address to share their respective list of assigned Group Identifiers currently in use.

10.6. Cross-group Message Injection

A same endpoint is allowed to and would likely use the same public/private key pair in multiple OSCORE groups, possibly administered by different Group Managers.

When a sender endpoint sends a message protected in pairwise mode to a recipient endpoint in an OSCORE group, a malicious group member may attempt to inject the message to a different OSCORE group also including the same endpoints (see Section 10.6.1).

This practically relies on altering the content of the OSCORE option, and having the same MAC in the ciphertext still correctly validating, which has a success probability depending on the size of the MAC.

As discussed in Section 10.6.2, the attack is practically infeasible if the message is protected in group mode, since the counter signature is bound also to the OSCORE option, through the Additional Authenticated Data used in the signing process (see Section 4.3.2).

10.6.1. Attack Description

Let us consider:

- Two OSCORE groups G1 and G2, with ID Context (Group ID) Gid1 and Gid2, respectively. Both G1 and G2 use the AEAD cipher AES-CCM-16-64-128, i.e. the MAC of the ciphertext is 8 bytes in size.

- A sender endpoint X which is member of both G1 and G2, and uses the same public/private key pair in both groups. The endpoint X has Sender ID Sid1 in G1 and Sender ID Sid2 in G2. The pairs (Sid1, Gid1) and (Sid2, Gid2) identify the same public key of X in G1 and G2, respectively.

- A recipient endpoint Y which is member of both G1 and G2, and uses the same public/private key pair in both groups. The endpoint Y has Sender ID Sid3 in G1 and Sender ID Sid4 in G2. The pairs (Sid3, Gid1) and (Sid4, Gid2) identify the same public key of Y in G1 and G2, respectively.

- A malicious endpoint Z is also member of both G1 and G2. Hence, Z is able to derive the symmetric keys associated to X in G1 and G2.

When X sends a message M1 addressed to Y in G1 and protected in pairwise mode, Z can intercept M1, and forge a valid message M2 to be injected in G2, making it appear as still sent by X to Y and valid to be accepted.

More in detail, Z intercepts and stops message M1, and forges a message M2 by changing the value of the OSCORE option from M1 as follows: the 'kid context' is changed from G1 to G2; and the 'kid' is changed from Sid1 to Sid2. Then, Z injects message M2 as addressed to Y in G2.

Upon receiving M2, there is a probability equal to 2^-64 that Y successfully verifies the same unchanged MAC by using Sid2 as 'request_kid' and using the Pairwise Recipient Key associated to X in G2.

Note that Z does not know the pairwise keys of X and Y, since it does not know and is not able to compute their shared Diffie-Hellman secret. Therefore, Z is not able to check offline if a performed forgery is actually valid, before sending the forged message to G2.

10.6.2. Attack Prevention in Group Mode

When a Group OSCORE message is protected with the group mode, the counter signature is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process (see Section 4.3.2).

That is, the countersignature is computed also over: the ID Context (Group ID) and the Partial IV, which are always present in group requests; as well as the Sender ID of the message originator, which is always present in all group requests and responses.

Since the signing process takes as input also the ciphertext of the COSE_Encrypt0 object, the countersignature is bound not only to the intended OSCORE group, hence to the triplet (Master Secret, Master Salt, ID Context), but also to a specific Sender ID in that group and to its specific symmetric key used for AEAD encryption, hence to the quartet (Master Secret, Master Salt, ID Context, Sender ID).

This makes it practically infeasible to perform the attack described in Section 10.6.1, since it would require the adversary to additionally forge a valid countersignature that replaces the original one in the forged message M2.

If the countersignature did not cover the OSCORE option, the attack would be possible also in group mode, since the same unchanged countersignature from messsage M1 would be also valid in message M2. Also, the following attack simplifications would hold, since Z is able to derive the Sender/Recipient Keys of X and Y in G1 and G2.

- If M2 is used as a request, Z can check offline if a performed forgery is actually valid before sending the forged message to G2.

That is, this attack would have a complexity of 2^64 offline calculations.

- If M2 is used as a response, Z can also change the response Partial IV, until the same unchanged MAC is successfully verified by using Sid2 as 'request_kid' and the symmetric key associated to X in G2. Since the Partial IV is 5 bytes in size, this requires 2^40 operations to test all the Partial IVs, which can be done in real-time. Also, the probability that a single given message M1 can be used to forge a response M2 for a given request would be equal to 2^-24, since there are more MAC values (8 bytes in size) than Partial IV values (5 bytes in size).

  Note that, by changing the Partial IV as discussed above, any member of G1 would also be able to forge a valid signed response message M2 to be injected in G1.

10.7. Group OSCORE for Unicast Requests

If a request is intended to be sent over unicast as addressed to a single group member, it is NOT RECOMMENDED for the client to protect the request by using the group mode as defined in Section 8.1.

This does not include the case where the client sends a request over unicast to a proxy, to be forwarded to multiple intended recipients over multicast [I-D.ietf-core-groupcomm-bis]. In this case, the client MUST protect the request with the group mode, even though it is sent to the proxy over unicast (see Section 8).

If the client uses the group mode with its own Sender Key to protect a unicast request to a group member, an on-path adversary can, right then or later on, redirect that request to one/many different group member(s) over unicast, or to the whole OSCORE group over multicast. By doing so, the adversary can induce the target group member(s) to perform actions intended for one group member only. Note that the adversary can be external, i.e. (s)he does not need to also be a member of the OSCORE group.

This is due to the fact that the client is not able to indicate the single intended recipient in a way which is secure and possible to process for Group OSCORE on the server side. In particular, Group OSCORE does not protect network addressing information such as the IP address of the intended recipient server. It follows that the server(s) receiving the redirected request cannot assert whether that was the original intention of the client, and would thus simply assume so.

The impact of such an attack depends especially on the REST method of the request, i.e. the Inner CoAP Code of the OSCORE request message. In particular, safe methods such as GET and FETCH would trigger (several) unintended responses from the targeted server(s), while not resulting in destructive behavior. On the other hand, non safe methods such as PUT, POST and PATCH/iPATCH would result in the target server(s) taking active actions on their resources and possible cyber-physical environment, with the risk of destructive consequences and possible implications for safety.

A client can instead use the pairwise mode as defined in Section 9.2, in order to protect a request sent to a single group member by using pairwise keying material (see Section 2.3). This prevents the attack discussed above by construction, as only the intended server is able to derive the pairwise keying material used by the client to protect the request. A client supporting the pairwise mode SHOULD use it to protect requests sent to a single group member over unicast, instead of using the group mode. For an example where this is not fulfilled, see Section 6.2.1 in [I-D.tiloca-core-observe-multicast-notifications].

With particular reference to block-wise transfers [RFC7959], Section 2.3.6 of [I-D.ietf-core-groupcomm-bis] points out that, while an initial request including the CoAP Block2 option can be sent over multicast, any other request in a transfer has to occur over unicast, individually addressing the servers in the group.

Additional considerations are discussed in Appendix E.3, with respect to requests including a CoAP Echo Option [I-D.ietf-core-echo-request-tag] that has to be sent over unicast, as a challenge-response method for servers to achieve synchronization of clients' Sender Sequence Number.

10.8. End-to-end Protection

The same considerations from Section 12.1 of [RFC8613] hold for Group OSCORE.

Additionally, (D)TLS and Group OSCORE can be combined for protecting message exchanges occurring over unicast. However, it is not possible to combine (D)TLS and Group OSCORE for protecting message exchanges where messages are (also) sent over multicast.

10.9. Master Secret

Group OSCORE derives the Security Context using the same construction as OSCORE, and by using the Group Identifier of a group as the related ID Context. Hence, the same required properties of the

Security Context parameters discussed in Section 3.3 of [RFC8613] hold for this document.

With particular reference to the OSCORE Master Secret, it has to be kept secret among the members of the respective OSCORE group and the Group Manager responsible for that group. Also, the Master Secret must have a good amount of randomness, and the Group Manager can generate it offline using a good random number generator. This includes the case where the Group Manager rekeys the group by generating and distributing a new Master Secret. Randomness requirements for security are described in [RFC4086].

10.10. Replay Protection

As in OSCORE, also Group OSCORE relies on sender sequence numbers included in the COSE message field 'Partial IV' and used to build AEAD nonces.

Note that the Partial IV of an endpoint does not necessarily grow monotonically. For instance, upon exhaustion of the endpoint Sender Sequence Number, the Partial IV also gets exhausted. As discussed in Section 2.4.3, this results either in the endpoint being individually rekeyed and getting a new Sender ID, or in the establishment of a new Security Context in the group. Therefore, uniqueness of (key, nonce) pairs (see Section 10.2) is preserved also when a new Security Context is established.

As discussed in Section 6.1, an endpoint that has just joined a group is exposed to replay attack, as it is not aware of the Sender Sequence Numbers currently used by other group members. Appendix E describes how endpoints can synchronize with others' Sender Sequence Number.

Unless exchanges in a group rely only on unicast messages, Group OSCORE cannot be used with reliable transport. Thus, unless only unicast messages are sent in the group, it cannot be defined that only messages with sequence numbers that are equal to the previous sequence number + 1 are accepted.

The processing of response messages described in Section 2.3.1 of [I-D.ietf-core-groupcomm-bis] also ensures that a client accepts a single valid response to a given request from each replying server, unless CoAP observation is used.

10.11. Client Aliveness

As discussed in Section 12.5 of [RFC8613], a server may use the CoAP Echo Option [I-D.ietf-core-echo-request-tag] to verify the aliveness of the client that originated a received request. This would also allow the server to (re-)synchronize with the client's Sender Sequence Number, as well as to ensure that the request is fresh and has not been replayed or (purposely) delayed, if it is the first one received from that client after having joined the group or rebooted (see Appendix E.3).

10.12. Cryptographic Considerations

The same considerations from Section 12.6 of [RFC8613] about the maximum Sender Sequence Number hold for Group OSCORE.

As discussed in Section 2.4.2, an endpoint that experiences an exhaustion of its own Sender Sequence Numbers MUST NOT protect further messages including a Partial IV, until it has derived a new Sender Context. This prevents the endpoint to reuse the same AEAD nonces with the same Sender Key.

In order to renew its own Sender Context, the endpoint SHOULD inform the Group Manager, which can either renew the whole Security Context by means of group rekeying, or provide only that endpoint with a new Sender ID value. In either case, the endpoint derives a new Sender Context, and in particular a new Sender Key.

Additionally, the same considerations from Section 12.6 of [RFC8613] hold for Group OSCORE, about building the AEAD nonce and the secrecy of the Security Context parameters.

The EdDSA signature algorithm and the elliptic curve Ed25519 [RFC8032] are mandatory to implement. For endpoints that support the pairwise mode, the ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs] and the X25519 curve [RFC7748] are also mandatory to implement.

Constrained IoT devices may alternatively represent Montgomery curves and (twisted) Edwards curves [RFC7748] in the short-Weierstrass form Wei25519, with which the algorithms ECDSA25519 and ECDH25519 can be used for signature operations and Diffie-Hellman secret calculation, respectively [I-D.ietf-lwig-curve-representations].

For many constrained IoT devices, it is problematic to support more than one signature algorithm or multiple whole cipher suites. As a consequence, some deployments using, for instance, ECDSA with NIST

P-256 may not support the mandatory signature algorithm but that should not be an issue for local deployments.

The derivation of pairwise keys defined in Section 2.3.1 is compatible with ECDSA and EdDSA asymmetric keys, but is not compatible with RSA asymmetric keys. The security of using the same key pair for Diffie-Hellman and for signing is demonstrated in [Degabriele].

10.13. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] hold for Group OSCORE.

10.14. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations. In particular, options are processed according to the same class U/I/E that they have for OSCORE. Therefore, the same privacy considerations from Section 12.8 of [RFC8613] hold for Group OSCORE.

Furthermore, the following privacy considerations hold, about the OSCORE option that may reveal information on the communicating endpoints.

o The 'kid' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, may reveal information about the Sender Endpoint. Since both requests and responses always include the 'kid' parameter, this may reveal information about both a client sending a group request and all the possibly replying servers sending their own individual response.

o The 'kid context' parameter, which is intended to help a recipient endpoint to find the right Security Context, reveals information about the sender endpoint. In particular, it reveals that the sender endpoint is a member of a particular OSCORE group, whose current Group ID is indicated in the 'kid context' parameter.

When receiving a group request, each of the recipient endpoints can reply with a response that includes its Sender ID as 'kid' parameter. All these responses will be matchable with the request through the Token. Thus, even if these responses do not include a 'kid context' parameter, it becomes possible to understand that the responder endpoints are in the same group of the requester endpoint.

Furthermore, using the mechanisms described in Appendix E.3 to achieve sequence number synchronization with a client may reveal when a server device goes through a reboot. This can be mitigated by the server device storing the precise state of the replay window of each known client on a clean shutdown.

Finally, the mechanism described in Section 10.5 to prevent collisions of Group Identifiers from different Group Managers may reveal information about events in the respective OSCORE groups. In particular, a Group Identifier changes when the corresponding group is rekeyed. Thus, Group Managers might use the shared list of Group Identifiers to infer the rate and patterns of group membership changes triggering a group rekeying, e.g. due to newly joined members or evicted (compromised) members. In order to alleviate this privacy concern, it should be hidden from the Group Managers which exact Group Manager has currently assigned which Group Identifiers in its OSCORE groups.

11. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[This Document]" with the RFC number of this specification and delete this paragraph.

This document has the following actions for IANA.

11.1. OSCORE Flag Bits Registry

IANA is asked to add the following value entry to the "OSCORE Flag Bits" subregistry defined in Section 13.7 of [RFC8613] as part of the "CoRE Parameters" registry.

Bit Position	Name	Description	Reference
2	Group Flag	Set to 1 if the message is protected with the group mode of Group OSCORE	[This Document]

12. References

12.1. Normative References

[COSE.Algorithms]
IANA, "COSE Algorithms", <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>.

[COSE.Key.Types]
IANA, "COSE Key Types", <https://www.iana.org/assignments/cose/cose.xhtml#key-type>.

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-02 (work in progress), November 2020.

[I-D.ietf-cose-countersign]
Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", draft-ietf-cose-countersign-01 (work in progress), October 2020.

[I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.

[I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-14 (work in progress), September 2020.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

12.2. Informative References

[Degabriele] Degabriele, J., Lehmann, A., Paterson, K., Smart, N., and M. Strefler, "On the Joint Security of Encryption and Signature in EMV", December 2011, <https://eprint.iacr.org/2011/615>.

[I-D.ietf-ace-key-groupcomm] Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-ietf-ace-key-groupcomm-10 (work in progress), November 2020.

[I-D.ietf-ace-key-groupcomm-oscore] Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-09 (work in progress), November 2020.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-35 (work in progress), June 2020.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-10 (work in progress), July 2020.

[I-D.ietf-lwig-curve-representations]
Struik, R., "Alternative Elliptic Curve Representations", draft-ietf-lwig-curve-representations-12 (work in progress), August 2020.

[I-D.ietf-lwig-security-protocol-comparison]
Mattsson, J., Palombini, F., and M. Vucinic, "Comparison of CoAP Security Protocols", draft-ietf-lwig-security-protocol-comparison-04 (work in progress), March 2020.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-38 (work in progress), May 2020.

[I-D.mattsson-cfrg-det-sigs-with-noise]
Mattsson, J., Thormarker, E., and S. Ruohomaa, "Deterministic ECDSA and EdDSA Signatures with Additional Randomness", draft-mattsson-cfrg-det-sigs-with-noise-02 (work in progress), March 2020.

[I-D.somaraju-ace-multicast]
Somaraju, A., Kumar, S., Tschofenig, H., and W. Werner, "Security for Low-Latency Group Communication", draft-somaraju-ace-multicast-02 (work in progress), October 2016.

[I-D.tiloca-core-observe-multicast-notifications]
Tiloca, M., Hoeglund, R., Amsuess, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", draft-tiloca-core-observe-multicast-notifications-04 (work in progress), November 2020.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document. The rest of this section refers to three types of groups:

- Application group, i.e. a set of CoAP endpoints that share a common pool of resources.

- Security group, as defined in Section 1.1 of this specification. There can be a one-to-one or a one-to-many relation between security groups and application groups, and vice versa.

- CoAP group, as defined in [I-D.ietf-core-groupcomm-bis] i.e. a set of CoAP endpoints, where each endpoint is configured to receive CoAP multicast requests that are sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups. There can be a one-to-one or a one-to-many relation between application groups and CoAP groups. Note that a device sending a CoAP request to a CoAP group is not necessarily itself a member of that group: it is a member only if it also has a CoAP server endpoint listening to requests for this CoAP group, sent to the associated IP multicast address and port. In order to provide secure group communication, all members of a CoAP group as

well as all further endpoints configured only as clients sending CoAP (multicast) requests to the CoAP group have to be member of a security group. There can be a one-to-one or a one-to-many relation between security groups and CoAP groups, and vice versa.

A.1. Assumptions

The following assumptions are assumed to be already addressed and are out of the scope of this document.

- Multicast communication topology: this document considers both 1-to-N (one sender and multiple recipients) and M-to-N (multiple senders and multiple recipients) communication topologies. The 1-to-N communication topology is the simplest group communication scenario that would serve the needs of a typical Low-power and Lossy Network (LLN). Examples of use cases that benefit from secure group communication are provided in Appendix B.

  In a 1-to-N communication model, only a single client transmits data to the CoAP group, in the form of request messages; in an M-to-N communication model (where M and N do not necessarily have the same value), M clients transmit data to the CoAP group. According to [I-D.ietf-core-groupcomm-bis], any possible proxy entity is supposed to know about the clients and to not perform aggregation of response messages from multiple servers. Also, every client expects and is able to handle multiple response messages associated to a same request sent to the CoAP group.

- Group size: security solutions for group communication should be able to adequately support different and possibly large security groups. The group size is the current number of members in a security group. In the use cases mentioned in this document, the number of clients (normally the controlling devices) is expected to be much smaller than the number of servers (i.e. the controlled devices). A security solution for group communication that supports 1 to 50 clients would be able to properly cover the group sizes required for most use cases that are relevant for this document. The maximum group size is expected to be in the range of 2 to 100 devices. Security groups larger than that should be divided into smaller independent groups.

- Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, also when not registered as a member of the security group.

- Provisioning and management of Security Contexts: a Security Context must be established among the members of the security

group. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, communication policies and security parameters in the security group. The actual provisioning and management of the Security Context is out of the scope of this document.

- Multicast data security ciphersuite: all members of a security group must agree on a ciphersuite to provide authenticity, integrity and confidentiality of messages in the group. The ciphersuite is specified as part of the Security Context.

- Backward security: a new device joining the security group should not have access to any old Security Contexts used before its joining. This ensures that a new member of the security group is not able to decrypt confidential data sent before it has joined the security group. The adopted key management scheme should ensure that the Security Context is updated to ensure backward confidentiality. The actual mechanism to update the Security Context and renew the group keying material in the security group upon a new member's joining has to be defined as part of the group key management scheme.

- Forward security: entities that leave the security group should not have access to any future Security Contexts or message exchanged within the security group after their leaving. This ensures that a former member of the security group is not able to decrypt confidential data sent within the security group anymore. Also, it ensures that a former member is not able to send protected messages to the security group anymore. The actual mechanism to update the Security Context and renew the group keying material in the security group upon a member's leaving has to be defined as part of the group key management scheme.

A.2. Security Objectives

The approach described in this document aims at fulfilling the following security objectives:

- Data replay protection: group request messages or response messages replayed within the security group must be detected.

- Data confidentiality: messages sent within the security group shall be encrypted.

- Group-level data confidentiality: the group mode provides group-level data confidentiality since messages are encrypted at a group level, i.e. in such a way that they can be decrypted by any member

of the security group, but not by an external adversary or other external entities.

- Pairwise data confidentiality: the pairwise mode especially provides pairwise data confidentiality, since messages are encrypted using pairwise keying material shared between any two group members, hence they can be decrypted only by the intended single recipient.

- Source message authentication: messages sent within the security group shall be authenticated. That is, it is essential to ensure that a message is originated by a member of the security group in the first place, and in particular by a specific, identifiable member of the security group.

- Message integrity: messages sent within the security group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with, either by a group member, or by an external adversary or other external entities which are not members of the security group.

- Message ordering: it must be possible to determine the ordering of messages coming from a single sender. In accordance with OSCORE [RFC8613], this results in providing absolute freshness of responses that are not notifications, as well as relative freshness of group requests and notification responses. It is not required to determine ordering of messages from different senders.

Appendix B. List of Use Cases

Group Communication for CoAP [I-D.ietf-core-groupcomm-bis] provides the necessary background for multicast-based CoAP communication, with particular reference to low-power and lossy networks (LLNs) and resource constrained environments. The interested reader is encouraged to first read [I-D.ietf-core-groupcomm-bis] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication, and refers to the three types of groups from Appendix A. Specific security requirements for these use cases are discussed in Appendix A.

- Lighting control: consider a building equipped with IP-connected lighting devices, switches, and border routers. The lighting devices acting as servers are organized into application groups and CoAP groups, according to their physical location in the building. For instance, lighting devices in a room or corridor can be configured as members of a single application group and corresponding CoAP group. Those ligthing devices together with the switches acting as clients in the same room or corridor can be

configured as members of the corresponding security group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in the CoAP group, while border routers connected to an IP network backbone (which is also multicast-enabled) can be used to interconnect routers in the building. Consequently, this would also enable logical groups to be formed even if devices with a role in the lighting application may be physically in different subnets (e.g. on wired and wireless networks). Connectivity between lighting devices may be realized, for instance, by means of IPv6 and (border) routers supporting 6LoWPAN [RFC4944][RFC6282]. Group communication enables synchronous operation of a set of connected lights, ensuring that the light preset (e.g. dimming level or color) of a large set of luminaires are changed at the same perceived time. This is especially useful for providing a visual synchronicity of light effects to the user. As a practical guideline, events within a 200 ms interval are perceived as simultaneous by humans, which is necessary to ensure in many setups. Devices may reply back to the switches that issue on/off/dimming commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status. In a typical lighting control scenario, a single switch is the only entity responsible for sending commands to a set of lighting devices. In more advanced lighting control use cases, a M-to-N communication topology would be required, for instance in case multiple sensors (presence or day-light) are responsible to trigger events to a set of lighting devices. Especially in professional lighting scenarios, the roles of client and server are configured by the lighting commissioner, and devices strictly follow those roles.

o Integrated building control: enabling Building Automation and Control Systems (BACSs) to control multiple heating, ventilation and air-conditioning units to pre-defined presets. Controlled units can be organized into application groups and CoAP groups in order to reflect their physical position in the building, e.g. devices in the same room can be configured as members of a single application group and corresponding CoAP group. As a practical guideline, events within intervals of seconds are typically acceptable. Controlled units are expected to possibly reply back to the BACS issuing control commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status.

o Software and firmware updates: software and firmware updates often comprise quite a large amount of data. This can overload a Low-power and Lossy Network (LLN) that is otherwise typically used to deal with only small amounts of data, on an infrequent base. Rather than sending software and firmware updates as unicast

messages to each individual device, multicasting such updated data to a larger set of devices at once displays a number of benefits. For instance, it can significantly reduce the network load and decrease the overall time latency for propagating this data to all devices. Even if the complete whole update process itself is secured, securing the individual messages is important, in case updates consist of relatively large amounts of data. In fact, checking individual received data piecemeal for tampering avoids that devices store large amounts of partially corrupted data and that they detect tampering hereof only after all data has been received. Devices receiving software and firmware updates are expected to possibly reply back, in order to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.

- o Parameter and configuration update: by means of multicast communication, it is possible to update the settings of a set of similar devices, both simultaneously and efficiently. Possible parameters are related, for instance, to network load management or network access controls. Devices receiving parameter and configuration updates are expected to possibly reply back, to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.

- o Commissioning of Low-power and Lossy Network (LLN) systems: a commissioning device is responsible for querying all devices in the local network or a selected subset of them, in order to discover their presence, and be aware of their capabilities, default configuration, and operating conditions. Queried devices displaying similarities in their capabilities and features, or sharing a common physical location can be configured as members of a single application group and corresponding CoAP group. Queried devices are expected to reply back to the commissioning device, in order to notify their presence, and provide the requested information and their current operational status.

- o Emergency multicast: a particular emergency related information (e.g. natural disaster) is generated and multicast by an emergency notifier, and relayed to multiple devices. The latter may reply back to the emergency notifier, in order to provide their feedback and local information related to the ongoing emergency. This kind of setups should additionally rely on a fault tolerance multicast algorithm, such as Multicast Protocol for Low-Power and Lossy Networks (MPL).

Appendix C. Example of Group Identifier Format

This section provides an example of how the Group Identifier (Gid) can be specifically formatted. That is, the Gid can be composed of two parts, namely a Group Prefix and a Group Epoch.

For each group, the Group Prefix is constant over time and is uniquely defined in the set of all the groups associated to the same Group Manager. The choice of the Group Prefix for a given group's Security Context is application specific. The size of the Group Prefix directly impact on the maximum number of distinct groups under the same Group Manager.

The Group Epoch is set to 0 upon the group's initialization, and is incremented by 1 each time new keying material, together with a new Gid, is distributed to the group in order to establish a new Security Context (see Section 3.1).

As an example, a 3-byte Gid can be composed of: i) a 1-byte Group Prefix '0xb1' interpreted as a raw byte string; and ii) a 2-byte Group Epoch interpreted as an unsigned integer ranging from 0 to 65535. Then, after having established the Common Context 61532 times in the group, its Gid will assume value '0xb1f05c'.

Using an immutable Group Prefix for a group assumes that enough time elapses before all possible Group Epoch values are used, since the Group Manager does not reassign the same Gid to the same group. Thus, the expected highest rate for addition/removal of group members and consequent group rekeying should be taken into account for a proper dimensioning of the Group Epoch size.

As discussed in Section 10.5, if endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible that Group Identifiers of different groups coincide at some point in time. In this case, a recipient has to handle coinciding Group Identifiers, and has to try using different Security Contexts to process an incoming message, until the right one is found and the message is correctly verified. Therefore, it is favourable that Group Identifiers from different Group Managers have a size that result in a small probability of collision. How small this probability should be is up to system designers.

Appendix D. Set-up of New Endpoints

An endpoint joins a group by explicitly interacting with the responsible Group Manager. When becoming members of a group, endpoints are not required to know how many and what endpoints are in the same group.

Communications between a joining endpoint and the Group Manager rely on the CoAP protocol and must be secured. Specific details on how to secure communications between joining endpoints and a Group Manager are out of the scope of this document.

The Group Manager must verify that the joining endpoint is authorized to join the group. To this end, the Group Manager can directly authorize the joining endpoint, or expect it to provide authorization evidence previously obtained from a trusted entity. Further details about the authorization of joining endpoints are out of scope.

In case of successful authorization check, the Group Manager generates a Sender ID assigned to the joining endpoint, before proceeding with the rest of the join process. That is, the Group Manager provides the joining endpoint with the keying material and parameters to initialize the Security Context (see Section 2). The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

It is RECOMMENDED that the join process adopts the approach described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

Appendix E. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by server endpoints to synchronize with Sender Sequence Numbers of client endpoints sending group requests.

The Group Manager MAY indicate which of such approaches are used in the group, as part of the group communication policies signalled to candidate group members upon their group joining.

If a server has recently lost the mutable Security Context, e.g. due to a reboot, the server has also to establish an updated Security Context before resuming to send protected messages to the group (see Section 2.4.1). Since this results in deriving a new Sender Key for its Sender Context, the server does not reuse the same pair (key, nonce), even when using the Partial IV of (old re-injected) requests to build the AEAD nonce for protecting the corresponding responses.

E.1. Best-Effort Synchronization

Upon receiving a group request from a client, a server does not take any action to synchronize with the Sender Sequence Number of that client. This provides no assurance at all as to message freshness, which can be acceptable in non-critical use cases.

With the notable exception of Observe notifications and responses following a group rekeying, it is optional for the server to use its Sender Sequence Number as Partial IV when protecting a response. Instead, for efficiency reasons, the server may rather use the request's Partial IV when protecting a response to that request.

E.2. Baseline Synchronization

Upon receiving a group request from a given client for the first time, a server initializes the last-seen Sender Sequence Number associated to that client in its corresponding Recipient Context. The server may also drop the group request without delivering it to the application. This method provides a reference point to identify if future group requests from the same client are fresher than the last one received.

A replay time interval exists, between when a possibly replayed or delayed message is originally transmitted by a given client and the first authentic fresh message from that same client is received. This can be acceptable for use cases where servers admit such a trade-off between performance and assurance of message freshness.

With the notable exception of Observe notifications and responses following a group rekeying, it is optional for the server to use its Sender Sequence Number as Partial IV when protecting a response. Instead, for efficiency reasons, the server may rather use the request's Partial IV when protecting a response to that request.

E.3. Challenge-Response Synchronization

A server performs a challenge-response exchange with a client, by using the Echo Option for CoAP described in Section 2 of [I-D.ietf-core-echo-request-tag] and according to Appendix B.1.2 of [RFC8613].

That is, upon receiving a group request from a particular client for the first time, the server processes the message as described in this specification, but, even if valid, does not deliver it to the application. Instead, the server replies to the client with an OSCORE protected 4.01 (Unauthorized) response message, including only the Echo Option and no diagnostic payload. The server MUST NOT set the Echo Option to a value which is both predictable and reusable. Since this response is protected with the Security Context used in the group, the client will consider the response valid upon successfully decrypting and verifying it.

The server stores the Echo Option value included therein, together with the pair (gid,kid), where 'gid' is the Group Identifier of the

OSCORE group and 'kid' is the Sender ID of the client in the group, as specified in the 'kid context' and 'kid' fields of the OSCORE Option of the group request, respectively. After a group rekeying has been completed and a new Security Context has been established in the group, which results also in a new Group Identifier (see Section 3.1), the server MUST delete all the stored Echo values associated to members of that group.

Upon receiving a 4.01 (Unauthorized) response that includes an Echo Option and originates from a verified group member, a client sends a request as a unicast message addressed to the same server, echoing the Echo Option value. The client MUST NOT send the request including the Echo Option over multicast.

In particular, the client does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This makes it possible for the client to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. In either case, the client uses a fresh Sender Sequence Number value from its own Sender Context. If the client stores group requests for possible retransmission with the Echo Option, it should not store a given request for longer than a pre-configured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed as a message, since the 'kid context' field including the Group Identifier of the OSCORE group is still present in the OSCORE Option as part of the COSE object (see Section 4).

Upon receiving the unicast request including the Echo Option, the server performs the following verifications.

o If the server does not store an Echo Option value for the pair (gid,kid), it considers: i) the time t1 when it has established the Security Context used to protect the received request; and ii) the time t2 when the request has been received. Since a valid request cannot be older than the Security Context used to protect it, the server verifies that (t2 - t1) is less than the largest amount of time acceptable to consider the request fresh.

o If the server stores an Echo Option value for the pair (gid,kid) associated to that same client in the same group, the server verifies that the option value equals that same stored value previously sent to that client.

If the verifications above fail, the server MUST NOT process the request further and MAY send a 4.01 (Unauthorized) response including an Echo Option.

In case of positive verification, the request is further processed and verified. Finally, the server updates the Recipient Context associated to that client, by setting the Replay Window according to the Sender Sequence Number from the unicast request conveying the Echo Option. The server either delivers the request to the application if it is an actual retransmission of the original one, or discards it otherwise. Mechanisms to signal whether the resent request is a full retransmission of the original one are out of the scope of this specification.

A server should not deliver group requests from a given client to the application until one valid request from that same client has been verified as fresh, as conveying an echoed Echo Option [I-D.ietf-core-echo-request-tag]. Also, a server may perform the challenge-response described above at any time, if synchronization with Sender Sequence Numbers of clients is (believed to be) lost, for instance after a device reboot. A client has to be always ready to perform the challenge-response based on the Echo Option in case a server starts it.

It is the role of the server application to define under what circumstances Sender Sequence Numbers lose synchronization. This can include experiencing a "large enough" gap D = (SN2 - SN1), between the Sender Sequence Number SN1 of the latest accepted group request from a client and the Sender Sequence Number SN2 of a group request just received from that client. However, a client may send several unicast requests to different group members as protected with the pairwise mode (see Section 9.2), which may result in the server experiencing the gap D in a relatively short time. This would induce the server to perform more challenge-response exchanges than actually needed.

To ameliorate this, the server may rather rely on a trade-off between the Sender Sequence Number gap D and a time gap T = (t2 - t1), where t1 is the time when the latest group request from a client was accepted and t2 is the time when the latest group request from that client has been received, respectively. Then, the server can start a challenge-response when experiencing a time gap T larger than a given, pre-configured threshold. Also, the server can start a challenge-response when experiencing a Sender Sequence Number gap D greater than a different threshold, computed as a monotonically increasing function of the currently experienced time gap T.

The challenge-response approach described in this appendix provides an assurance of absolute message freshness. However, it can result in an impact on performance which is undesirable or unbearable, especially in large groups where many endpoints at the same time might join as new members or lose synchronization.

Note that endpoints configured as silent servers are not able to perform the challenge-response described above, as they do not store a Sender Context to secure the 4.01 (Unauthorized) response to the client. Therefore, silent servers should adopt alternative approaches to achieve and maintain synchronization with sender sequence numbers of clients.

Since requests including the Echo Option are sent over unicast, a server can be a victim of the attack discussed in Section 10.7, when such requests are protected with the group mode of Group OSCORE, as described in Section 8.1.

Instead, protecting requests with the Echo Option by using the pairwise mode of Group OSCORE as described in Section 9.2 prevents the attack in Section 10.7. In fact, only the exact server involved in the Echo exchange is able to derive the correct pairwise key used by the client to protect the request including the Echo Option.

In either case, an internal on-path adversary would not be able to mix up the Echo Option value of two different unicast requests, sent by a same client to any two different servers in the group. In fact, if the group mode was used, this would require the adversary to forge the client's counter signature in both such requests. As a consequence, each of the two servers remains able to selectively accept a request with the Echo Option only if it is waiting for that exact integrity-protected Echo Option value, and is thus the intended recipient.

Appendix F. No Verification of Signatures in Group Mode

There are some application scenarios using group communication that have particularly strict requirements. One example of this is the requirement of low message latency in non-emergency lighting applications [I-D.somaraju-ace-multicast]. For those applications which have tight performance constraints and relaxed security requirements, it can be inconvenient for some endpoints to verify digital signatures in order to assert source authenticity of received messages protected with the group mode. In other cases, the signature verification can be deferred or only checked for specific actions. For instance, a command to turn a bulb on where the bulb is already on does not need the signature to be checked. In such situations, the counter signature needs to be included anyway as part of a message protected with the group mode, so that an endpoint that needs to validate the signature for any reason has the ability to do so.

In this specification, it is NOT RECOMMENDED that endpoints do not verify the counter signature of received messages protected with the

group mode. However, it is recognized that there may be situations where it is not always required. The consequence of not doing the signature validation in messages protected with the group mode is that security in the group is based only on the group-authenticity of the shared keying material used for encryption. That is, endpoints in the group would have evidence that the received message has been originated by a group member, although not specifically identifiable in a secure way. This can violate a number of security requirements, as the compromise of any element in the group means that the attacker has the ability to control the entire group. Even worse, the group may not be limited in scope, and hence the same keying material might be used not only for light bulbs but for locks as well. Therefore, extreme care must be taken in situations where the security requirements are relaxed, so that deployment of the system will always be done safely.

Appendix G. Example Values with COSE Capabilities

The table below provides examples of values for Counter Signature Parameters in the Common Context (see Section 2.1.3), for different values of Counter Signature Algorithm.

Counter Signature Algorithm	Example Values for Counter Signature Parameters
(-8) // EdDSA	[1], [1, 6] // 1: OKP ; 1: OKP, 6: Ed25519
(-8) // EdDSA	[1], [1, 6] // 1: OKP ; 1: OKP, 7: Ed448
(-7) // ES256	[2], [2, 1] // 2: EC2 ; 2: EC2, 1: P-256
(-35) // ES384	[2], [2, 2] // 2: EC2 ; 2: EC2, 2: P-384
(-36) // ES512	[2], [2, 3] // 2: EC2 ; 2: EC2, 3: P-512
(-37) // PS256	[], [3] // empty ; 3: RSA
(-38) // PS384	[], [3] // empty ; 3: RSA
(-39) // PS512	[], [3] // empty ; 3: RSA

Figure 4: Examples of Counter Signature Parameters

The table below provides examples of values for Secret Derivation Parameters in the Common Context (see Section 2.1.5), for different values of Secret Derivation Algorithm.

Secret Derivation Algorithm	Example Values for Secret Derivation Parameters
(-27) // ECDH-SS // + HKDF-256	[1], [1, 6] // 1: OKP ; 1: OKP, 4: X25519
(-27) // ECDH-SS // + HKDF-256	[1], [1, 6] // 1: OKP ; 1: OKP, 5: X448
(-27) // ECDH-SS // + HKDF-256	[2], [2, 1] // 2: EC2 ; 2: EC2, 1: P-256
(-27) // ECDH-SS // + HKDF-256	[2], [2, 2] // 2: EC2 ; 2: EC2, 2: P-384
(-27) // ECDH-SS // + HKDF-256	[2], [2, 3] // 2: EC2 ; 2: EC2, 3: P-512

Figure 5: Examples of Secret Derivation Parameters

The table below provides examples of values for the 'par_countersign_key' element of the 'algorithms' array used in the two external_aad structures (see Section 4.3.1 and Section 4.3.2), for different values of Counter Signature Algorithm.

Counter Signature Algorithm	Example Values for 'par_countersign_key'
(-8) // EdDSA	[1, 6] // 1: OKP , 6: Ed25519
(-8) // EdDSA	[1, 6] // 1: OKP , 7: Ed448
(-7) // ES256	[2, 1] // 2: EC2 , 1: P-256
(-35) // ES384	[2, 2] // 2: EC2 , 2: P-384
(-36) // ES512	[2, 3] // 2: EC2 , 3: P-512
(-37) // PS256	[3] // 3: RSA
(-38) // PS384	[3] // 3: RSA
(-39) // PS512	[3] // 3: RSA

Figure 6: Examples of 'par_countersign_key'

Appendix H. **Document Updates**

RFC EDITOR: PLEASE REMOVE THIS SECTION.

H.1. **Version -09 to -10**

- o Removed 'Counter Signature Key Parameters' from the Common Context.

- New parameters in the Common Context covering the DH secret derivation.

- New counter signature header parameter from draft-ietf-cose-countersign.

- Stronger policies non non-recycling of Sender IDs and Gid.

- The Sender Sequence Number is reset when establishing a new Security Context.

- Added 'request_kid_context' in the aad_array.

- The server can respond with 5.03 if the client's public key is not available.

- The observer client stores an invariant identifier of the group.

- Relaxed storing of original 'kid' for observer clients.

- Both client and server store the 'kid_context' of the original observation request.

- The server uses a fresh PIV if protecting the response with a Security Context different from the one used to protect the request.

- Clarifications on MTI algorithms and curves.

- Removed optimized requests.

- Overall clarifications and editorial revision.

H.2. Version -08 to -09

- Pairwise keys are discarded after group rekeying.

- Signature mode renamed to group mode.

- The parameters for countersignatures use the updated COSE registries. Newly defined IANA registries have been removed.

- Pairwise Flag bit renamed as Group Flag bit, set to 1 in group mode and set to 0 in pairwise mode.

- Dedicated section on updating the Security Context.

- By default, sender sequence numbers and replay windows are not reset upon group rekeying.
- An endpoint implementing only a silent server does not support the pairwise mode.
- Separate section on general message reception.
- Pairwise mode moved to the document body.
- Considerations on using the pairwise mode in non-multicast settings.
- Optimized requests are moved as an appendix.
- Normative support for the signature and pairwise mode.
- Revised methods for synchronization with clients' sender sequence number.
- Appendix with example values of parameters for countersignatures.
- Clarifications and editorial improvements.

H.3. Version -07 to -08

- Clarified relation between pairwise mode and group communication (Section 1).
- Improved definition of "silent server" (Section 1.1).
- Clarified when a Recipient Context is needed (Section 2).
- Signature checkers as entities supported by the Group Manager (Section 2.3).
- Clarified that the Group Manager is under exclusive control of Gid and Sender ID values in a group, with Sender ID values under each Gid value (Section 2.3).
- Mitigation policies in case of recycled 'kid' values (Section 2.4).
- More generic exhaustion (not necessarily wrap-around) of sender sequence numbers (Sections 2.5 and 10.11).
- Pairwise key considerations, as to group rekeying and Sender Sequence Numbers (Section 3).

- Added reference to static-static Diffie-Hellman shared secret (Section 3).

- Note for implementation about the external_aad for signing (Sectino 4.3.2).

- Retransmission by the application for group requests over multicast as Non-Confirmable (Section 7).

- A server MUST use its own Partial IV in a response, if protecting it with a different context than the one used for the request (Section 7.3).

- Security considerations: encryption of pairwise mode as alternative to group-level security (Section 10.1).

- Security considerations: added approach to reduce the chance of global collisions of Gid values from different Group Managers (Section 10.5).

- Security considerations: added implications for block-wise transfers when using the signature mode for requests over unicast (Section 10.7).

- Security considerations: (multiple) supported signature algorithms (Section 10.13).

- Security considerations: added privacy considerations on the approach for reducing global collisions of Gid values (Section 10.15).

- Updates to the methods for synchronizing with clients' sequence number (Appendix E).

- Simplified text on discovery services supporting the pairwise mode (Appendix G.1).

- Editorial improvements.

H.4. Version -06 to -07

- Updated abstract and introduction.

- Clarifications of what pertains a group rekeying.

- Derivation of pairwise keying material.

- Content re-organization for COSE Object and OSCORE header compression.

- Defined the Pairwise Flag bit for the OSCORE option.

- Supporting CoAP Observe for group requests and responses.

- Considerations on message protection across switching to new keying material.

- New optimized mode based on pairwise keying material.

- More considerations on replay protection and Security Contexts upon key renewal.

- Security considerations on Group OSCORE for unicast requests, also as affecting the usage of the Echo option.

- Clarification on different types of groups considered (application/security/CoAP).

- New pairwise mode, using pairwise keying material for both requests and responses.

H.5. Version -05 to -06

- Group IDs mandated to be unique under the same Group Manager.

- Clarifications on parameter update upon group rekeying.

- Updated external_aad structures.

- Dynamic derivation of Recipient Contexts made optional and application specific.

- Optional 4.00 response for failed signature verification on the server.

- Removed client handling of duplicated responses to multicast requests.

- Additional considerations on public key retrieval and group rekeying.

- Added Group Manager responsibility on validating public keys.

- Updates IANA registries.

- Reference to RFC 8613.

- Editorial improvements.

H.6. Version -04 to -05

- Added references to draft-dijk-core-groupcomm-bis.

- New parameter Counter Signature Key Parameters (Section 2).

- Clarification about Recipient Contexts (Section 2).

- Two different external_aad for encrypting and signing (Section 3.1).

- Updated response verification to handle Observe notifications (Section 6.4).

- Extended Security Considerations (Section 8).

- New "Counter Signature Key Parameters" IANA Registry (Section 9.2).

H.7. Version -03 to -04

- Added the new "Counter Signature Parameters" in the Common Context (see Section 2).

- Added recommendation on using "deterministic ECDSA" if ECDSA is used as counter signature algorithm (see Section 2).

- Clarified possible asynchronous retrieval of keying material from the Group Manager, in order to process incoming messages (see Section 2).

- Structured Section 3 into subsections.

- Added the new 'par_countersign' to the aad_array of the external_aad (see Section 3.1).

- Clarified non reliability of 'kid' as identity indicator for a group member (see Section 2.1).

- Described possible provisioning of new Sender ID in case of Partial IV wrap-around (see Section 2.2).

- The former signature bit in the Flag Byte of the OSCORE option value is reverted to reserved (see Section 4.1).

- Updated examples of compressed COSE object, now with the sixth less significant bit in the Flag Byte of the OSCORE option value set to 0 (see Section 4.3).

- Relaxed statements on sending error messages (see Section 6).

- Added explicit step on computing the counter signature for outgoing messages (see Setions 6.1 and 6.3).

- Handling of just created Recipient Contexts in case of unsuccessful message verification (see Sections 6.2 and 6.4).

- Handling of replied/repeated responses on the client (see Section 6.4).

- New IANA Registry "Counter Signature Parameters" (see Section 9.1).

H.8. Version -02 to -03

- Revised structure and phrasing for improved readability and better alignment with draft-ietf-core-object-security.

- Added discussion on wrap-Around of Partial IVs (see Section 2.2).

- Separate sections for the COSE Object (Section 3) and the OSCORE Header Compression (Section 4).

- The countersignature is now appended to the encrypted payload of the OSCORE message, rather than included in the OSCORE Option (see Section 4).

- Extended scope of Section 5, now titled " Message Binding, Sequence Numbers, Freshness and Replay Protection".

- Clarifications about Non-Confirmable messages in Section 5.1 "Synchronization of Sender Sequence Numbers".

- Clarifications about error handling in Section 6 "Message Processing".

- Compacted list of responsibilities of the Group Manager in Section 7.

- Revised and extended security considerations in Section 8.

- Added IANA considerations for the OSCORE Flag Bits Registry in Section 9.

- Revised Appendix D, now giving a short high-level description of a new endpoint set-up.

H.9. Version -01 to -02

- Terminology has been made more aligned with RFC7252 and draft-ietf-core-object-security: i) "client" and "server" replace the old "multicaster" and "listener", respectively; ii) "silent server" replaces the old "pure listener".

- Section 2 has been updated to have the Group Identifier stored in the 'ID Context' parameter defined in draft-ietf-core-object-security.

- Section 3 has been updated with the new format of the Additional Authenticated Data.

- Major rewriting of Section 4 to better highlight the differences with the message processing in draft-ietf-core-object-security.

- Added Sections 7.2 and 7.3 discussing security considerations about uniqueness of (key, nonce) and collision of group identifiers, respectively.

- Minor updates to Appendix A.1 about assumptions on multicast communication topology and group size.

- Updated Appendix C on format of group identifiers, with practical implications of possible collisions of group identifiers.

- Updated Appendix D.2, adding a pointer to draft-palombini-ace-key-groupcomm about retrieval of nodes' public keys through the Group Manager.

- Minor updates to Appendix E.3 about Challenge-Response synchronization of sequence numbers based on the Echo option from draft-ietf-core-echo-request-tag.

H.10. Version -00 to -01

- Section 1.1 has been updated with the definition of group as "security group".

- Section 2 has been updated with:

  * Clarifications on etablishment/derivation of Security Contexts.

      * A table summarizing the the additional context elements compared to OSCORE.

   o Section 3 has been updated with:

      * Examples of request and response messages.

      * Use of CounterSignature0 rather than CounterSignature.

      * Additional Authenticated Data including also the signature algorithm, while not including the Group Identifier any longer.

   o Added Section 6, listing the responsibilities of the Group Manager.

   o Added Appendix A (former section), including assumptions and security objectives.

   o Appendix B has been updated with more details on the use cases.

   o Added Appendix C, providing an example of Group Identifier format.

   o Appendix D has been updated to be aligned with draft-palombini-ace-key-groupcomm.

Acknowledgments

The authors sincerely thank Christian Amsuess, Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Klaus Hartke, Rikard Hoeglund, Richard Kelsey, John Mattsson, Dave Robin, Jim Schaad, Ludwig Seitz, Peter van der Stok and Erik Thormarker for their feedback and comments.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; the H2020 project SIFIS-Home (Grant agreement 952652); the SSF project SEC4Factory under the grant RIT17-0032; and the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de