Workgroup: CoRE Working Group
Internet-Draft:
draft-ietf-core-oscore-groupcomm-14
Published: 7 March 2022
Intended Status: Standards Track
Expires: 8 September 2022
Authors: M. Tiloca RISE AB
G. Selander Ericsson AB
F. Palombini Ericsson AB
J. Mattsson Ericsson AB
J. Park Universitaet Duisburg-Essen

Group OSCORE - Secure Group Communication for CoAP

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. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive a symmetric pairwise key with any other member of the group for pairwise OSCORE communication.

Table of Contents

1. Introduction
  1.1. Terminology
2. Security Context
  2.1. Common Context
    2.1.1. AEAD Algorithm
    2.1.2. ID Context
    2.1.3. Group Manager Authentication Credential
    2.1.4. Signature Encryption Algorithm
    2.1.5. Signature Algorithm
    2.1.6. Group Encryption Key
    2.1.7. Pairwise Key Agreement Algorithm
  2.2. Sender Context and Recipient Context
  2.3. Authentication Credentials
  2.4. Pairwise Keys
    2.4.1. Derivation of Pairwise Keys
    2.4.2. ECDH with Montgomery Coordinates
    2.4.3. Usage of Sequence Numbers
    2.4.4. Security Context for Pairwise Mode
  2.5. Update of Security Context
    2.5.1. Loss of Mutable Security Context
    2.5.2. Exhaustion of Sender Sequence Number
    2.5.3. Retrieving New Security Context Parameters
3. The Group Manager
  3.1. Support for Additional Entities
  3.2. Management of Group Keying Material
    3.2.1. Recycling of Identifiers
  3.3. Responsibilities of the Group Manager
4. The COSE Object
  4.1. Countersignature
    4.1.1. Keystream Derivation
    4.1.2. Clarifications on Using a Countersignature
  4.2. The 'kid' and 'kid context' parameters
  4.3. external_aad
5. OSCORE Header Compression
  5.1. Examples of Compressed COSE Objects
    5.1.1. Examples in Group Mode
    5.1.2. Examples in Pairwise Mode

6. Message Binding, Sequence Numbers, Freshness and Replay Protection
  6.1. Supporting Observe
  6.2. Update of Replay Window
  6.3. Message Freshness
7. Message Reception
8. Message Processing in Group Mode
  8.1. Protecting the Request
    8.1.1. Supporting Observe
  8.2. Verifying the Request
    8.2.1. Supporting Observe
  8.3. Protecting the Response
    8.3.1. Supporting Observe
  8.4. Verifying the Response
    8.4.1. Supporting Observe
  8.5. External Signature Checkers
9. Message Processing in Pairwise Mode
  9.1. Pre-Conditions
  9.2. Main Differences from OSCORE
  9.3. Protecting the Request
  9.4. Verifying the Request
  9.5. Protecting the Response
  9.6. Verifying the Response
10. Challenge-Response Synchronization
11. Implementation Compliance
12. Security Considerations
  12.1. Security of the Group Mode
  12.2. Security of the Pairwise Mode
  12.3. Uniqueness of (key, nonce)
  12.4. Management of Group Keying Material
  12.5. Update of Security Context and Key Rotation
    12.5.1. Late Update on the Sender
    12.5.2. Late Update on the Recipient
  12.6. Collision of Group Identifiers
  12.7. Cross-group Message Injection
    12.7.1. Attack Description
    12.7.2. Attack Prevention in Group Mode
  12.8. Prevention of Group Cloning Attack
  12.9. Group OSCORE for Unicast Requests
  12.10. End-to-end Protection
  12.11. Master Secret
  12.12. Replay Protection
  12.13. Message Freshness
  12.14. Client Aliveness
  12.15. Cryptographic Considerations
  12.16. Message Segmentation
  12.17. Privacy Considerations
13. IANA Considerations
  13.1. OSCORE Flag Bits Registry

14. References
  14.1. Normative References
  14.2. Informative References
Appendix A. Assumptions and Security Objectives
  A.1. Assumptions
  A.2. Security Objectives
Appendix B. List of Use Cases
Appendix C. Example of Group Identifier Format
Appendix D. Set-up of New Endpoints
Appendix E. Document Updates
  E.1. Version -13 to -14
  E.2. Version -12 to -13
  E.3. Version -11 to -12
  E.4. Version -10 to -11
  E.5. Version -09 to -10
  E.6. Version -08 to -09
  E.7. Version -07 to -08
  E.8. Version -06 to -07
  E.9. Version -05 to -06
  E.10. Version -04 to -05
  E.11. Version -03 to -04
  E.12. Version -02 to -03
  E.13. Version -01 to -02
  E.14. Version -00 to -01
Acknowledgments
Authors' Addresses

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). Group communication for CoAP [I-D.ietf-core-groupcomm-bis] mainly uses UDP/IP multicast as the underlying data transport.

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 the transport layer 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, a security protocol for Group communication for CoAP [I-D.ietf-core-groupcomm-bis], 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. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive a symmetric pairwise key with any other member of the group for pairwise OSCORE communication. Just like OSCORE, Group OSCORE is independent of the transport layer and works wherever CoAP does.

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). This prevents observers from accessing addressing information conveyed in CoAP options that would not be protected by Group OSCORE, but would be protected by DTLS. These options include Uri-Host, Uri-Port and Proxy-Uri. Note that DTLS does not define how to secure messages sent over IP multicast.

Group OSCORE defines two modes of operation, that can be used independently or together:

* 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 signature algorithms as well as signature verification by intermediaries. This mode is defined in Section 8.

* In the pairwise mode, two group members exchange OSCORE requests and responses (typically) 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.

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 endpoints, it is desirable to restrict the amount of secret keying material provided to each endpoint. 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). However, it has the benefit of providing a single shared secret, while distributing only the public keys of group members or a subset of those. 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 [RFC8949]; COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and related countersignatures [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.

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

* Authentication credential: set of information associated with an entity, including that entity's public key and parameters associated with the public key. Examples of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert]. Further details about authentication credentials are provided in Section 2.3.

* Group: a set of endpoints that share group keying material and security parameters (Common Context, see Section 2). That is, unless otherwise specified, the term group used in this document refers 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".

* 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 3.3.

* 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.

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

* Birth Gid: with respect to a group member, the Gid obtained by that group member upon (re-)joining the group.

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

* Key Generation Number: an integer value identifying the current version of the keying material used in a group.

* 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

As per the terminology in Section 1.1, this document refers to a group as a set of endpoints sharing keying material and security parameters for executing the Group OSCORE protocol. Each endpoint of a group is aware of whether the group uses the group mode, or the pairwise mode, or both. Then, an endpoint can use any mode it supports if also used in the group.

All members of a group maintain a Security Context as defined in Section 3 of [RFC8613] and extended as defined in this section. How the Security Context is established by the group members is out of scope for this document, 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, including the Security Context, is described in terms of a Group Manager (see Section 3). In particular, the Group Manager indicates whether the group uses the group mode, the pairwise mode, or both of them, as part of the group data provided to candidate group members when joining the group.

The remainder of this section provides further details about the Security Context of Group OSCORE. In particular, 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).

* One Common Context, shared by all the endpoints in the group. Several new parameters are included in the Common Context.

  If a Group Manager is used for maintaining the group, the Common Context is extended with the authentication credential of the Group Manager, including the Group Manager's public key. When processing messages, the authentication credential of the Group Manager is included in the external additional authenticated data (see Section 4.3).

  If the group uses the group mode, the Common context is extended with the following new parameters.

  - Signature Encryption Algorithm and Signature Algorithm. These relate to the encryption/decryption operations and to the computation/verification of countersignatures, respectively,

when a message is protected with the group mode (see Section 8).

- Group Encryption Key, used to perform encryption/decryption of countersignatures, when a message is protected with the group mode (see Section 8).

If the group uses the pairwise mode, the Common Context is extended with a Pairwise Key Agreement Algorithm used for agreement on a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.4.1) to protect messages with the pairwise mode (see Section 9).

* One Sender Context, extended with the endpoint's private key and authentication credential including the endpoint's public key.

  The private key is used to sign messages protected with the group mode, or for deriving pairwise keys in pairwise mode (see Section 2.4). The authentication credential is used for deriving pairwise keys in pairwise mode, and is included in the external additional authenticated data when processing outgoing messages (see Section 9).

  If the endpoint supports the pairwise mode, the Sender Context is also extended with the Pairwise Sender Keys associated with the other endpoints (see Section 2.4).

  The Sender Context is omitted if the endpoint is configured exclusively as silent server.

* One Recipient Context for each other endpoint from which messages are received. It is not necessary to maintain Recipient Contexts associated with endpoints from which messages are not (expected to be) received. The Recipient Context is extended with the authentication credential of the other endpoint, including that endpoint's public key.

  The public key is used to verify the signature of messages protected with the group mode from the other endpoint and for deriving the pairwise keys in pairwise mode (see Section 2.4). The authentication credential is used for deriving pairwise keys in pairwise mode, and is included in the external additional authenticated data when processing incoming messages from the other endpoint (see Section 9).

  If the endpoint supports the pairwise mode, then the Recipient Context is also extended with the Pairwise Recipient Key associated with the other endpoint (see Section 2.4).

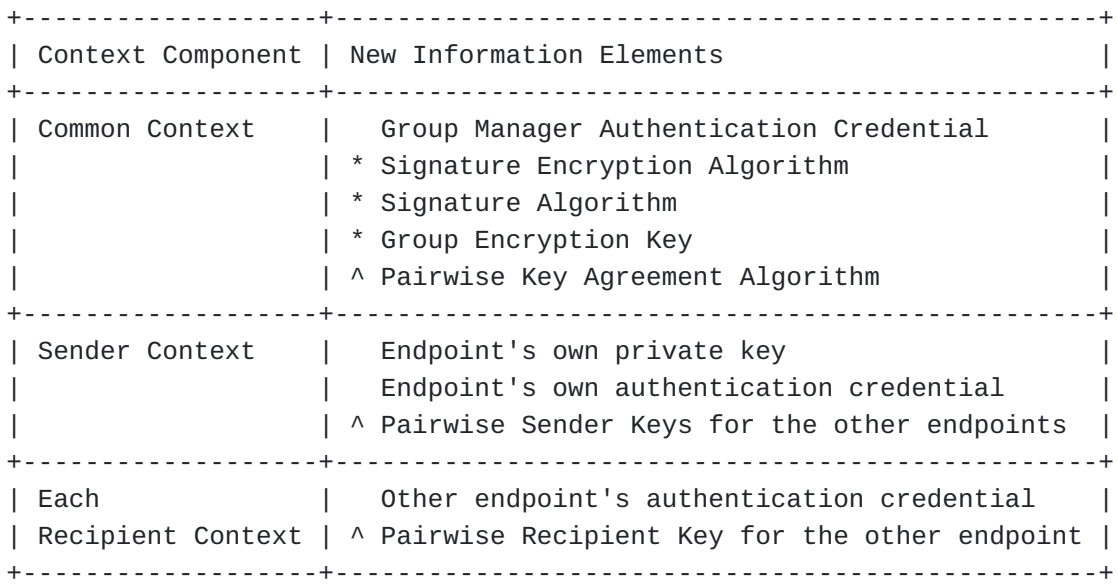

Figure 1: Additions to the OSCORE Security Context. The optional elements labeled with * (with ^) are present only if the group uses the group mode (the pairwise mode).

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. AEAD Algorithm

AEAD Algorithm identifies the COSE AEAD algorithm to use for encryption, when messages are protected using the pairwise mode (see Section 9). This algorithm MUST provide integrity protection. This parameter is immutable once the Common Context is established, and it is not relevant if the group uses only the group mode.

2.1.2. ID Context

The ID Context parameter (see Sections 3.1 and 3.3 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 12.6).

2.1.3. Group Manager Authentication Credential

Group Manager Authentication Credential specifies the authentication credential of the Group Manager, including the Group Manager's public key. This is included in the external additional authenticated data when processing messages (see Section 4.3).

Each group member MUST obtain the authentication credential of the Group Manager with a valid proof-of-possession of the corresponding private key, for instance from the Group Manager itself when joining the group. Further details on the provisioning of the Group Manager's authentication credential to the group members are out of the scope of this document.

2.1.4. Signature Encryption Algorithm

Signature Encryption Algorithm identifies the algorithm to use for encryption, when messages are protected using the group mode (see Section 8). This algorithm MAY provide integrity protection. This parameter is immutable once the Common Context is established.

This algorithm is not used to encrypt the countersignature in messages protected using the group mode, for which the method defined in Section 4.1 is used.

2.1.5. Signature Algorithm

Signature Algorithm identifies the digital signature algorithm used to compute a countersignature 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.

2.1.6. Group Encryption Key

Group Encryption Key specifies the encryption key for deriving a keystream to encrypt/decrypt a countersignature, when a message is protected with the group mode (see Section 8).

The Group Encryption Key is derived as defined for Sender/Recipient Keys in Section 3.2.1 of [RFC8613], with the following differences.

* The 'id' element of the 'info' array is the empty byte string.

* The 'alg_aead' element of the 'info' array takes the value of Signature Encryption Algorithm from the Common Context (see Section 2.1.5).

* The 'type' element of the 'info' array is "Group Encryption Key". The label is an ASCII string and does not include a trailing NUL byte.

* L and the 'L' element of the 'info' array are the size of the key for the Signature Encryption Algorithm from the Common Context (see Section 2.1.5), in bytes.

2.1.7. Pairwise Key Agreement Algorithm

Pairwise Key Agreement 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 2.4.1) to protect messages with the pairwise mode (see Section 9). This parameter is immutable once the Common Context is established.

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]).

The derivation of Sender/Recipient Keys and Common IV defined in OSCORE applies also to Group OSCORE, with the following extensions compared to Section 3.2.1 of [RFC8613].

* If the group uses (also) the group mode, the 'alg_aead' element of the 'info' array takes the value of Signature Encryption Algorithm from the Common Context (see Section 2.1.5).

* If the group uses only the pairwise mode, the 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context (see Section 2.1.1).

The Sender ID SHALL be unique for each endpoint in a group with a certain tuple (Master Secret, Master Salt, 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 key of the sender and the authentication credential including the corresponding public key can, for example, be generated by the endpoint or provisioned during manufacturing.

With the exception of the authentication credential of the sender endpoint and the possibly associated pairwise keys, a receiver endpoint can derive a complete Security Context from a received Group OSCORE message and the Common Context. The authentication credentials in the Recipient Contexts can be retrieved from the Group Manager (see Section 3) upon joining the group. An authentication credential 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 authentication credential. Such devices can be configured to drop a received message for which there is no (complete) Recipient Context, and retrieve the sender endpoint's authentication credential in order to have it available to verify subsequent messages from that endpoint.

An endpoint admits a maximum amount of Recipient Contexts for a same Security Context, e.g., due to memory limitations. After reaching that limit, the creation of a new Recipient Context results in an overflow. When this happens, the endpoint has to delete a current Recipient Context to install the new one. It is up to the application to define policies for selecting the current Recipient Context to delete. If the new Recipient Context has been installed after the endpoint has experienced the overflow above, then the Recipient Context is initialized with an invalid Replay Window, and accordingly requires the endpoint to take appropriate actions (see Section 2.5.1.2).

2.3. Authentication Credentials

In a group, the following MUST hold for the authentication credential of each endpoint as well as for the authentication credential of the Group Manager.

* All authentication credentials MUST be encoded according to the same format used in the group. The used format MUST provide the public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).

* All authentication credentials and the public key specified therein MUST be for the public key algorithm used in the group and aligned with the possible associated parameters used in the group, e.g., the used elliptic curve (when applicable).

  If the group uses (also) the group mode, the public key algorithm is the Signature Algorithm used in the group. If the group uses only the pairwise mode, the public key algorithm is the Pairwise Key Agreement Algorithm used in the group.

  If the authentication credentials are X.509 certificates [RFC7925] or C509 certificates [I-D.ietf-cose-cbor-encoded-cert], the public key algorithm is fully described by the "algorithm" field of the "SubjectPublicKeyInfo" structure, and by the "subjectPublicKeyAlgorithm" element, respectively.

  If authentication credentials are CBOR Web Tokens (CWTs) or CWT Claims Sets (CCSs) [RFC8392], the public key algorithm is fully described by a COSE key type and its "kty" and "crv" parameters.

Authentication credentials are used to derive pairwise keys (see Section 2.4.1) and are included in the external additional authenticated data when processing messages (see Section 4.3). In both these cases, an endpoint in a group MUST treat authentication credentials as opaque data, i.e., by considering the same binary representation made available to other endpoints in the group, possibly through a designated trusted source (e.g., the Group Manager).

For example, an X.509 certificate is provided as its direct binary serialization. If C509 certificates or CWTs are used as authentication credentials, each is provided as the binary serialization of a (possibly tagged) CBOR array. If CCSs are used as authentication credentials, each is provided as the binary serialization of a CBOR map.

If authentication credentials are CWTs, then the untagged CWT associated with an entity is stored in the Security Context and used as authentication credential for that entity.

If authentication credentials are X.509 / C509 certificates or CWTs and the authentication credential associated with an entity is provided within a chain or a bag, then only the end-entity certificate or end-entity untagged CWT is stored in the Security Context and used as authentication credential for that entity.

Storing whole authentication credentials rather than only a subset of those may result in a non-negligible storage overhead. On the other hand, it also ensures that authentication credentials are correctly used in a simple, flexible and non-error-prone way, also taking into account future credential formats as entirely new or extending existing ones. In particular, it is ensured that:

* When used to derive pairwise keys and when included in the external additional authenticated data, authentication credentials can also specify possible metadata and parameters related to the included public key. Besides the public key algorithm, these comprise other relevant pieces of information such as key usage, expiration time, issuer and subject.

* All endpoints using another endpoint's authentication credential use exactly the same binary serialization, as obtained and distributed by the credential provider (e.g., the Group Manager) and as originally crafted by the credential issuer. In turn, this does not require to define and maintain canonical subsets of authentication credentials and their corresponding encoding, and spares endpoints from error-prone re-encoding operations.

Depending on the particular deployment and the intended group size, limiting the storage overhead of endpoints in a group can be an incentive for system/network administrators to prefer using a compact format of authentication credentials in the first place.

2.4. 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 defined in Section 9.

Group OSCORE keys used for both signature and encryption MUST be used only for purposes related to Group OSCORE. These include the processing of messages with Group OSCORE, as well as performing proof-of-possession of private keys, e.g., upon joining a group through the Group Manager (see Section 3).

2.4.1. 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 X in the group by means of the AEAD Algorithm from the Common Context (see Section 2.1.1). The key derivation of these so-called pairwise keys follows the same construction as in Section 3.2.1 of [RFC8613]:

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

with

```
IKM-Sender    = Sender Auth Cred | Recipient Auth Cred | Shared Secret
IKM-Recipient = Recipient Auth Cred | Sender Auth Cred | Shared Secret
```

where:

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

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

* HKDF is the OSCORE HKDF algorithm [RFC8613] from the Common Context.

* The Sender Key from the Sender Context is used as salt in the HKDF, when deriving the Pairwise Sender Key.

* The Recipient Key from the Recipient Context associated with endpoint X is used as salt in the HKDF, when deriving the Pairwise Recipient Key.

* Sender Auth Cred is the endpoint's own authentication credential from the Sender Context.

* Recipient Auth Cred is the endpoint X's authentication credential from the Recipient Context associated with the endpoint X.

* The Shared Secret is computed as a cofactor Diffie-Hellman shared secret, see Section 5.7.1.2 of [NIST-800-56A], using the Pairwise Key Agreement Algorithm. The endpoint uses its private key from the Sender Context and the other endpoint's public key included in Recipient Auth Cred. Note the requirement of validation of public keys in Section 12.15. For X25519 and X448, the procedure is described in Section 5 of [RFC7748] using public keys mapped to Montgomery coordinates, see Section 2.4.2.

* IKM-Sender is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Sender Key. IKM-Sender is the byte string concatenation of Sender Auth Cred, Recipient Auth Cred and the Shared Secret. The authentication credentials Sender Auth Cred and Recipient Auth Cred are binary encoded as defined in Section 2.3.

* IKM-Recipient is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Recipient Key. IKM-Recipient is the byte string concatenation of Recipient Auth Cred, Sender Auth Cred and the Shared Secret. The authentication credentials Recipient Auth Cred and Sender Auth Cred are binary encoded as defined in Section 2.3.

* info and L are as defined in Section 3.2.1 of [RFC8613]. That is:

  - The 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context (see Section 2.1.1).

  - L and the 'L' element of the 'info' array are the size of the key for the AEAD Algorithm from the Common Context (see Section 2.1.1), in bytes.

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]. For further details, see Section 2.4.2. ECC asymmetric keys in Montgomery or Weirstrass form are used directly in the key agreement algorithm without coordinate mapping.

After establishing a partially or completely new Security Context (see Section 2.5 and Section 3.2), 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.

2.4.2. ECDH with Montgomery Coordinates

2.4.2.1. Curve25519

The y-coordinate of the other endpoint's Ed25519 public key is decoded as specified in Section 5.1.3 of [RFC8032]. The Curve25519 u-coordinate is recovered as u = (1 + y) / (1 - y) (mod p) following the map in Section 4.1 of [RFC7748]. Note that the mapping is not defined for y = 1, and that y = -1 maps to u = 0 which corresponds to the neutral group element and thus will result in a degenerate shared secret. Therefore implementations MUST abort if the y-coordinate of the other endpoint's Ed25519 public key is 1 or -1 (mod p).

The private signing key byte strings (= the lower 32 bytes used for generating the public key, see step 1 of Section 5.1.5 of [RFC8032]) are decoded the same way for signing in Ed25519 and scalar multiplication in X25519. Hence, to compute the shared secret the endpoint applies the X25519 function to the Ed25519 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.4.2.2. Curve448

The y-coordinate of the other endpoint's Ed448 public key is decoded as specified in Section 5.2.3. of [RFC8032]. The Curve448 u-coordinate is recovered as u = y^2 * (d * y^2 - 1) / (y^2 - 1) (mod p) following the map from "edwards448" in Section 4.2 of [RFC7748], and also using the relation x^2 = (y^2 - 1)/(d * y^2 - 1) from the curve equation. Note that the mapping is not defined for y = 1 or -1. Therefore implementations MUST abort if the y-coordinate of the peer endpoint's Ed448 public key is 1 or -1 (mod p).

The private signing key byte strings (= the lower 57 bytes used for generating the public key, see step 1 of Section 5.2.5 of [RFC8032]) are decoded the same way for signing in Ed448 and scalar multiplication in X448. Hence, to compute the shared secret the endpoint applies the X448 function to the Ed448 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.4.3. 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. In turn, this contributes to a sooner exhaustion of the Sender Sequence Number space of the client, which would then require to take actions for deriving a new Sender Context before resuming communications in the group (see Section 2.5.2).

2.4.4. Security Context for Pairwise Mode

If the pairwise mode is supported, the Security Context additionally includes Pairwise Key Agreement Algorithm 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 2.4.1) 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 2.5.3). 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 with 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.

2.5. 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 2.5.3).

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 2.5.1) or exhaustion of Sender Sequence Numbers (see Section 2.5.2).

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 and MUST accordingly take the actions defined in Section 2.5.1.

2.5.1. Loss of Mutable Security Context

An endpoint may lose its mutable Security Context, e.g., due to a reboot (see Section 2.5.1.1) or to an overflow of Recipient Contexts (see Section 2.5.1.2).

In such a case, the endpoint needs to prevent the re-use of a nonce with the same AEAD key, and to handle incoming replayed messages.

2.5.1.1. Reboot and Total Loss

In case a loss of the Sender Context and/or of the Recipient Contexts is detected (e.g., following a reboot), the endpoint MUST NOT protect further messages using this Security Context to avoid reusing an AEAD nonce with the same AEAD key.

In particular, before resuming its operations in the group, the endpoint MUST retrieve new Security Context parameters from the Group Manager (see Section 2.5.3) and use them to derive a new Sender Context (see Section 2.2). Since this includes a newly derived Sender Key, a server will 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.

From then on, the endpoint MUST use the latest installed Sender Context to protect outgoing messages. Also, newly created Recipient Contexts will have a Replay Window which is initialized as valid.

If 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 and MUST NOT accept incoming messages from other group members, as currently unable to detect possible replays.

An adversary may leverage the above to perform a Denial of Service attack and prevent some group members from communicating altogether. That is, the adversary can first block the communication path between the Group Manager and some individual group members. This can be achieved, for instance, by injecting fake responses to DNS queries for the Group Manager hostname, or by removing a network link used for routing traffic towards the Group Manager. Then, the adversary can induce a reboot for some endpoints in the group, e.g., by triggering a short power outage. After that, such endpoints that have lost their Sender Context and/or Recipient Contexts following the reboot would not be able to obtain new Security Context parameters from the Group Manager, as specified above. Thus, they would not be able to further communicate in the group until connectivity with the Group Manager is restored.

2.5.1.2. Overflow of Recipient Contexts

After reaching the maximum amount of Recipient Contexts, an endpoint will experience an overflow when installing a new Recipient Context, as it requires to first delete an existing one (see Section 2.2).

Every time this happens, the Replay Window of the new Recipient Context is initialized as not valid. Therefore, the endpoint MUST take the following actions, before accepting request messages from the client associated with the new Recipient Context.

If it is not configured as silent server, the endpoint MUST either:

- Retrieve new Security Context parameters from the Group Manager and derive a new Sender Context, as defined in Section 2.5.1.1; or

- When receiving a first request to process with the new Recipient Context, use the approach specified in Section 10 and based on the Echo Option for CoAP [RFC9175], if supported. In particular, the endpoint MUST use its Partial IV when generating the AEAD nonce and MUST include the Partial IV in the response message conveying the Echo Option. If the endpoint supports the CoAP Echo Option, it is RECOMMENDED to take this approach.

If it is configured exclusively as silent server, the endpoint MUST wait for the next group rekeying to occur, in order to derive a new Security Context and re-initialize the Replay Window of each Recipient Contexts as valid.

2.5.2. 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 2.5.3), and use them to derive a new Sender Context (see Section 2.2).

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

2.5.3. 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 2.5.3.1); and ii) establishment of a new Security Context for the group (see Section 2.5.3.2). The update of the Replay Window in each of the Recipient Contexts is discussed in Section 6.2.

As group membership changes, or as group members get new Sender IDs (see Section 2.5.3.1) 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 identifier 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 authentication credentials and associated 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 an authentication credential with the public key included therein is

currently the one associated with a 'kid' value, after a number of consecutive failed verifications.

2.5.3.1. 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 not been used in the group since the latest time when the current Gid value was assigned to the group (see Section 3.2).

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 resets the Sender Sequence Number in its Sender Context 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.

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 2.5.2). An endpoint may request to re-join the group, e.g., because of losing its mutable Security Context (see Section 2.5.1), and is provided with 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 3.2).

2.5.3.2. New Security Context for the Group

The Group Manager may establish a new Security Context for the group (see Section 3.2). The Group Manager does not necessarily establish a new Security Context for the group if one member has an outdated Security Context (see Section 2.5.3.1), 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 its Sender Sequence Number in its Sender Context to 0.

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

  - For each ongoing observation where it is an observer client and that it wants to keep active, it resets to 0 the Notification Number of each associated server (see Section 6.1).

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.

*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 12.5.1).

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 12.5.

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 document is based on the existence of an entity called Group Manager and responsible for the group, but it does not mandate how the Group Manager interacts with the group members. The list of responsibilities of the Group Manager is compiled in Section 3.3.

A possible Group Manager to use is specified 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 an integer Key Generation Number to each of its groups, identifying the current version of the keying material used in that group. The first Key Generation Number assigned to every group MUST be 0. Separately for each group, the value of the Key Generation Number increases strictly monotonically, each time the Group Manager distributes new keying material to that group (see Section 3.2). That is, if the current Key Generation Number for a group is X, then X+1 will denote the keying material distributed and used in that group immediately after the current one.

The Group Manager assigns unique Group Identifiers (Gids) to the groups under its control. Also, for each group, the Group Manager assigns unique Sender IDs (and thus Recipient IDs) to the respective group members. According to a hierarchical approach, the Gid value assigned to a group is associated with a dedicated space for the values of Sender ID and Recipient ID of the members of that group. When an endpoint (re-)joins a group, it is provided also with the current Gid to use in the group.

The Group Manager maintains records of the authentication credentials of endpoints in a group, and provides information about the group and its members to other group members and to external entities with selected roles (see Section 3.1). Upon endpoints' joining, the Group Manager collects such authentication credentials 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 authentication credential to the Group Manager, for example upon joining the group.

Furthermore, when joining the group or later on as a group member, an endpoint can retrieve from the Group Manager the authentication credential of the Group Manager as well as the authentication credential and other information associated with other members of the group, with which it can derive the corresponding Recipient Context. Together with the requested authentication credentials, the Group Manager MUST provide the Sender ID of the associated group members and the current Key Generation Number in the group. 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.

3.1. Support for Additional Entities

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 authentication credentials of group members and other selected group data from the Group Manager, in order to solely verify countersignatures of messages protected in group mode (see Section 8.5).

In order to verify countersignatures of messages in a group, a signature checker needs to retrieve the following information about that group from the Group Manager.

- The current ID Context (Gid) used in the group.

- The authentication credentials of the group members and the authentication credential of the Group Manager.

  If the signature checker is provided with a CWT for a given entity, then the authentication credential associated with that entity that the signature checker stores and uses is the untagged CWT.

  If the signature checker is provided with a chain or a bag of X.509 / C509 certificates or of CWTs for a given entity, then the authentication credential associated with that entity that the signature checker stores and uses is just the end-entity certificate or end-entity untagged CWT.

- The current Group Encryption Key (see Section 2.1.6).

- The identifiers of the algorithms used in the group (see Section 2), i.e.: i) Signature Encryption Algorithm and Signature Algorithm; and ii) AEAD Algorithm and Pairwise Key Agreement Algorithm, if the group uses also the pairwise mode.

A signature checker MUST be authorized before it can retrieve such information. To this end, the same method mentioned above based on the ACE framework [I-D.ietf-ace-oauth-authz] can be used.

3.2. Management of Group Keying Material

In order to establish a new Security Context for a group, the Group Manager MUST generate and assign to the group a new Group Identifier (Gid) and a new value for the Master Secret parameter. When doing so, a new value for the Master Salt parameter MAY also be generated and assigned to the group. When establishing the new Security Context, the Group Manager should preserve the current value of the Sender ID of each group member.

The specific group key management scheme used to distribute new keying material is out of the scope of this document. A simple group key management scheme is defined in [I-D.ietf-ace-key-groupcomm-oscore]. When possible, the delivery of rekeying messages should use a reliable transport, in order to reduce chances of group members missing a rekeying instance.

The set of group members should not be assumed as fixed, i.e., the group membership is subject to changes, possibly on a frequent basis.

The Group Manager MUST rekey the group when one or more endpoints leave the group. An endpoint may leave the group at own initiative, or may be evicted from the group by the Group Manager, e.g., in case an endpoint is compromised, or is suspected to be compromised. In either case, rekeying the group excludes such endpoints from future communications in the group, and thus preserves forward security. If a network node is compromised or suspected to be compromised, the Group Manager MUST evict from the group all the endpoints hosted by that node that are member of the group and rekey the group accordingly.

If required by the application, the Group Manager MUST rekey the group also before one or more new joining endpoints are added to the group, thus preserving backward security.

The establishment of the new Security Context for the group takes the following steps.

1. The Group Manager MUST increment the Key Generation Number for the group by 1.

2. The Group Manager MUST build a set of stale Sender IDs including:

   * The Sender IDs that, during the current Gid, were both assigned to an endpoint and subsequently relinquished (see Section 2.5.3.1).

   * The current Sender IDs of the group members that the upcoming group rekeying aims to exclude from future group communications, if any.

3. The Group Manager rekeys the group, by distributing:

   * The new keying material, i.e., the new Master Secret, the new Gid and (optionally) the new Master Salt.

   * The new Key Generation Number from step 1.

*The set of stale Sender IDs from step 2.

Further information may be distributed, depending on the specific group key management scheme used in the group.

When receiving the new group keying materal, a group member considers the received stale Sender IDs and performs the following actions.

*The group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group.

*The group member MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

After that, the group member installs the new keying material and derives the corresponding new Security Context.

A group member might miss one group rekeying or more consecutive instances. As a result, the group member will retain old group keying material with Key Generation Number GEN_OLD. Eventually, the group member can notice the discrepancy, e.g., by repeatedly failing to verify incoming messages, or by explicitly querying the Group Manager for the current Key Generation Number. Once the group member gains knowledge of having missed a group rekeying, it MUST delete the old keying material it stores.

Then, the group member proceeds according to the following steps.

1. The group member retrieves from the Group Manager the current group keying material, together with the current Key Generation Number GEN_NEW. The group member MUST NOT install the obtained group keying material yet.

2. The group member asks the Group Manager for the set of stale Sender IDs.

3. If no exact indication can be obtained from the Group Manager, the group member MUST remove all the authentication credentials from its list of group members' authentication credentials used in the group and MUST delete all its Recipient Contexts used in the group.

   Otherwise, the group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group, and MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

4. The group member installs the current group keying material, and derives the corresponding new Security Context.

Alternatively, the group member can re-join the group. In such a case, the group member MUST take one of the following two actions.

* The group member performs steps 2 and 3 above. Then, the group member re-joins the group.

* The group member re-joins the group with the same roles it currently has in the group, and, during the re-joining process, it asks the Group Manager for the authentication credentials of all the current group members.

  Then, given Z the set of authentication credentials received from the Group Manager, the group member removes every authentication credential which is not in Z from its list of group members' authentication credentials used in the group, and deletes each of its Recipient Contexts used in the group that does not include any of the authentication credentials in Z.

By removing authentication credentials and deleting Recipient Contexts associated with stale Sender IDs, it is ensured that a recipient endpoint storing the latest group keying material does not store the authentication credentials of sender endpoints that are not current group members. This in turn allows group members to rely on stored authentication credentials to confidently assert the group membership of sender endpoints, when receiving incoming messages protected in group mode (see Section 8).

3.2.1. Recycling of Identifiers

This section specifies how the Group Manager handles and possibly reassigns Gid values and Sender ID values in a group.

3.2.1.1. Recycling of Group Identifiers

Since the Gid value changes every time a group is rekeyed, it can happen that, after several rekeying instances, the whole space of Gid values has been used for the group in question. When this happens, the Group Manager has no available Gid values to use that have never been assigned to the group during the group's lifetime.

The occurrence of such an event and how long it would take to occur depend on the format and encoding of Gid values used in the group (see, e.g., Appendix C), as well as on the frequency of rekeying instances yielding a change of Gid value. Independently for each

group under its control, the Group Manager can take one of the two following approaches.

* The Group Manager does not reassign Gid values. That is, once the whole space of Gid values has been used for a group, the Group Manager terminates the group and may re-establish a new group.

* While the Gid value changes every time a group is rekeyed, the Group Manager can reassign Gid values previously used during a group's lifetime. By doing so, the group can continue to exist even once the whole space of Gid values has been used.

  The Group Manager MAY support and use this approach. In such a case, the Group Manager MUST take additional actions when handling Gid values and rekeying the group, as specified below.

  When a node (re-)joins the group and it is provided with the current Gid to use in the group, the Group Manager considers such a Gid as the Birth Gid of that endpoint for that group. For each group member, the Group Manager MUST store the latest corresponding Birth Gid until that member leaves the group. In case the endpoint has in fact re-joined the group, the newly determined Birth Gid overwrites the one currently stored.

  When establishing a new Security Context for the group, the Group Manager takes the additional following step between steps 1 and 2 of Section 3.2.

  A. The Group Manager MUST check if the new Gid to be distributed is equal to the Birth Gid of any of the current group members. If any of such "elder members" is found in the group, then:

    - The Group Manager MUST evict the elder members from the group. That is, the Group Manager MUST terminate their membership and MUST rekey the group in such a way that the new keying material is not provided to those evicted elder members.

      This ensures that an Observe notification [RFC7641] can never successfully match against the Observe requests of two different observations. In fact, the excluded elder members would eventually re-join the group, thus terminating any of their ongoing (long-lasting) observations (see Section 6.1). Therefore, it is ensured by construction that no observer client can have two different ongoing observations such that the two respective Observe requests were protected using the same Partial IV, Gid and Sender ID.

    - Until a further following group rekeying, the Group Manager MUST store the list of those latest-evicted elder members. If any of those endpoints re-joins the group before a further

following group rekeying occurs, the Group Manager MUST NOT rekey the group upon their re-joining. When one of those endpoints re-joins the group, the Group Manager can rely, e.g., on the ongoing secure communication association to recognize the endpoint as included in the stored list.

3.2.1.2. Recycling of Sender IDs

From the moment when a Gid is assigned to a group until the moment a new Gid is assigned to that same group, the Group Manager MUST NOT reassign a Sender ID within the group. This prevents to reuse a Sender ID ('kid') with the same Gid, Master Secret and Master Salt. Within this restriction, the Group Manager can assign a Sender ID used under an old Gid value (including under a same, recycled Gid value), thus avoiding Sender ID values to irrecoverably grow in size.

Even when an endpoint joining a group is recognized as a current member of that group, e.g., through the ongoing secure communication association, the Group Manager MUST assign a new Sender ID different than the one currently used by the endpoint in the group, unless the group is rekeyed first and a new Gid value is established.

3.2.1.3. Relation between Identifiers and Keying Material

Figure 2 overviews the different identifiers and keying material components, considering their relation and possible reuse across group rekeying.

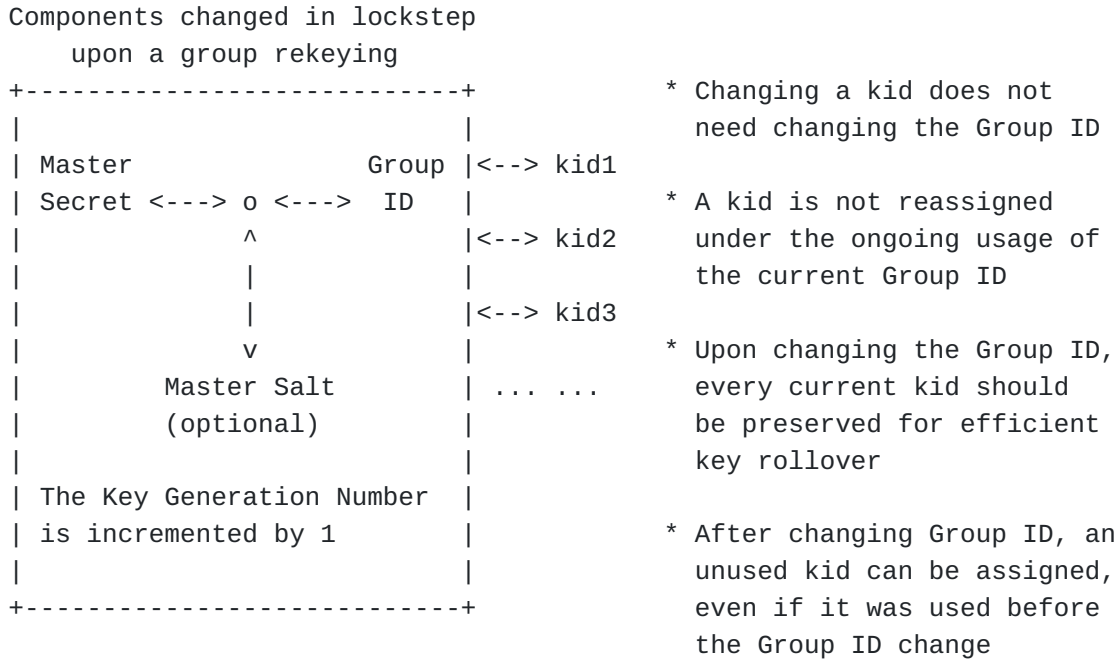

Figure 2: Relations among keying material components.

3.3. 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, ensuring uniqueness of Gids within the set of its OSCORE groups and, optionally, the secure recycling of Gids.

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. Updating the Key Generation Number and the Gid of its OSCORE groups, upon renewing the respective Security Context.

6. 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 or re-joining. This includes ensuring that:

   * Each Sender ID is unique within each of the OSCORE groups;

   * Each Sender ID is not reassigned within the same group since the latest time when the current Gid value was assigned to the group. That is, the Sender ID is not reassigned even to a current group member re-joining the same group, without a rekeying happening first.

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

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

9. 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.

10. Assisting a group member that has missed a group rekeying instance to understand which authentication credentials and

Recipient Contexts to delete, as associated with former group members.

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

12. Validating that the format and parameters of authentication credentials of group members are consistent with the public key algorithm and related parameters used in the respective OSCORE group.

The Group Manager specified 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. Countersignature

When protecting a message in group mode, the 'unprotected' field MUST additionally include the following parameter:

* COSE_CounterSignature0: its value is set to the encrypted countersignature of the COSE object, namely ENC_SIGNATURE. That is:

  - The countersignature of the COSE object, namely SIGNATURE, is 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 Signature Algorithm in the Security Context.

    In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 4.3 of this document, and the ciphertext of the COSE object as payload.

  - The encrypted countersignature, namely ENC_SIGNATURE, is computed as

ENC_SIGNATURE = SIGNATURE XOR KEYSTREAM

where KEYSTREAM is derived as per Section 4.1.1.

4.1.1. Keystream Derivation

The following defines how an endpoint derives the keystream KEYSTREAM, used to encrypt/decrypt the countersignature of an outgoing/incoming message M protected in group mode.

The keystream SHALL be derived as follows, by using the HKDF Algorithm from the Common Context (see Section 3.2 of [RFC8613]), which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

KEYSTREAM = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows.

- salt takes as value the Partial IV (PIV) used to protect M. Note that, if M is a response, salt takes as value either: i) the fresh Partial IV generated by the server and included in the response; or ii) the same Partial IV of the request generated by the client and not included in the response.

- IKM is the Group Encryption Key from the Common Context (see Section 2.1.6).

- info is the serialization of a CBOR array consisting of (the notation follows [RFC8610]):

```
info = [
  id : bstr,
  id_context : bstr,
  type : bool,
  L: uint
]
```

where:

- id is the Sender ID of the endpoint that generated PIV.

- id_context is the ID Context (Gid) used when protecting M.

  Note that, in case of group rekeying, a server might use a different Gid when protecting a response, compared to the Gid that it used to verify (that the client used to protect) the request, see Section 8.3.

*type is the CBOR simple value "true" (0xf5) if M is a request, or the CBOR simple value "false" (0xf4) otherwise.

*L is the size of the countersignature, as per Signature Algorithm from the Common Context (see Section 2.1.5), in bytes.

4.1.2. Clarifications on Using a Countersignature

Note that the literature commonly refers to a countersignature as a signature computed by an entity A over a document already protected by a different entity B.

However, the COSE_Countersignature0 structure belongs to the set of abbreviated countersignatures defined in Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign], which were designed primarily to deal with the problem of encrypted group messaging, but where it is required to know who originated the message.

Since the parameters for computing or verifying the abbreviated countersignature generated by A are provided by the same context used to describe the security processing performed by B and to be countersigned, these structures are applicable also when the two entities A and B are actually the same one, like the sender of a Group OSCORE message protected in group mode.

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, if the request was protected in group mode. That is, unlike in [RFC8613], the 'kid' parameter is always present in responses to a request that was protected in group mode.

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 [RFC8613], and is defined in this section.

The same external_aad structure is used in group mode and pairwise mode for authenticated encryption/decryption (see Section 5.3 of [I-D.ietf-cose-rfc8152bis-struct]), as well as in group mode for computing and verifying the countersignature (see Section 4.4 of [I-D.ietf-cose-rfc8152bis-struct]).

In particular, the external_aad includes also the Signature Algorithm, the Signature Encryption Algorithm, the Pairwise Key Agreement Algorithm, the value of the 'kid context' in the COSE object of the request, the OSCORE option of the protected message, the sender's authentication credential, and the Group Manager's authentication credential.

The external_aad SHALL be a CBOR array wrapped in a bstr object as defined below, following the notation of [RFC8610]:

```
external_aad = bstr .cbor aad_array

aad_array = [
   oscore_version : uint,
   algorithms : [alg_aead : int / tstr / null,
                 alg_signature_enc : int / tstr / null,
                 alg_signature : int / tstr / null,
                 alg_pairwise_key_agreement : int / tstr / null],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr,
   request_kid_context : bstr,
   OSCORE_option: bstr,
   sender_cred: bstr,
   gm_cred: bstr / null
]
```

Figure 3: external_aad

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

*The 'algorithms' array is extended as follows.

The parameter 'alg_aead' MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the pairwise mode, regardless whether the endpoint supports the pairwise mode or not. Otherwise, this parameter MUST encode the value of AEAD Algorithm from the Common Context (see Section 2.1.1), as per Section 5.4 of [RFC8613].

Furthermore, the 'algorithms' array additionally includes:

- 'alg_signature_enc', which specifies Signature Encryption Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the group mode, regardless whether the endpoint supports the group mode or not. Otherwise, this parameter MUST encode the value of Signature Encryption

Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this AEAD algorithm.

-'alg_signature', which specifies Signature Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the group mode, regardless whether the endpoint supports the group mode or not. Otherwise, this parameter MUST encode the value of Signature Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this signature algorithm.

-'alg_pairwise_key_agreement', which specifies Pairwise Key Agreement Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the pairwise mode, regardless whether the endpoint supports the pairwise mode or not. Otherwise, this parameter MUST encode the value of Pairwise Key Agreement Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this HKDF algorithm.

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

In case Observe [RFC7641] is used, this enables endpoints to safely keep an observation active beyond a possible change of Gid (i.e., of ID Context), following a group rekeying (see Section 3.2). In fact, it ensures that every notification cryptographically matches with only one observation request, rather than with multiple ones that were protected with different keying material but share the same 'request_kid' and 'request_piv' values.

*The new element 'OSCORE_option', containing the value of the OSCORE Option present in the protected message, encoded as a binary string. This prevents the attack described in Section 12.7 when using the group mode, as further explained in Section 12.7.2.

Note for implementation: this construction requires the OSCORE option of the message to be generated and finalized before computing the ciphertext of the COSE_Encrypt0 object (when using the group mode or the pairwise mode) and before calculating the countersignature (when using the group mode). Also, the aad_array needs to be large enough to contain the largest possible OSCORE option.

* The new element 'sender_cred', containing the sender's authentication credential. This parameter MUST be set to a CBOR byte string, which encodes the sender's authentication credential in its original binary representation made available to other endpoints in the group (see Section 2.3).

* The new element 'gm_cred', containing the Group Manager's authentication credential. If no Group Manager maintains the group, this parameter MUST encode the CBOR simple value "null" (0xf6). Otherwise, this parameter MUST be set to a CBOR byte string, which encodes the Group Manager's authentication credential in its original binary representation made available to other endpoints in the group (see Section 2.3). This prevents the attack described in Section 12.8.

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 document defines the usage of the sixth least significant bit, called "Group Flag", in the first byte of the OSCORE option containing the OSCORE flag bits. This flag bit is specified in Section 13.1.

* 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, the COSE_CounterSignature0 byte string as described in Section 4 is assumed to be 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:h'de9e ... f1' },
h'aea0155667924dff8a24e4cb35b9'
]
```

* After compression (85 bytes):

Flag byte: 0b00111001 = 0x39 (1 byte)

Option Value: 0x39 05 03 44 61 6c 25 (7 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 de9e ... f1
(14 bytes + size of the encrypted countersignature)

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

* Before compression (88 bytes):

```
[
h'',
{ 4:h'52', 11:h'ca1e ... b3' },
h'60b035059d9ef5667c5a0710823b'
]
```

* After compression (80 bytes):

  Flag byte: 0b00101000 = 0x28 (1 byte)

  Option Value: 0x28 52 (2 bytes)

  Payload: 0x60b035059d9ef5667c5a0710823b ca1e ... b3
  (14 bytes + size of the encrypted countersignature)

5.1.2. Examples in Pairwise Mode

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

* Before compression (29 bytes):

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

* After compression (21 bytes):

  Flag byte: 0b00011001 = 0x19 (1 byte)

  Option Value: 0x19 05 03 44 61 6c 25 (7 bytes)

  Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

*Response with ciphertext = 0x60b035059d9ef5667c5a0710823b and no Partial IV.

* Before compression (18 bytes):

```
[
h'',
{},
h'60b035059d9ef5667c5a0710823b'
]
```

* After compression (14 bytes):

  Flag byte: 0b00000000 = 0x00 (1 byte)

  Option Value: 0x (0 bytes)

  Payload: 0x60b035059d9ef5667c5a0710823b (14 bytes)

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

The requirements and properties described in Section 7 of [RFC8613] also apply to Group OSCORE. 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. In addition, the following holds for Group OSCORE.

6.1. Supporting Observe

When Observe [RFC7641] is used, a client maintains for each ongoing observation one Notification Number for each different server. Then, separately for each server, the client uses the associated Notification Number to perform ordering and replay protection of notifications received from that server (see Section 8.4.1).

Group OSCORE allows to preserve an observation active indefinitely, even in case the group is rekeyed, with consequent change of ID Context, or in case the observer client obtains a new Sender ID.

As defined in Section 8 when discussing support for Observe, this is achieved by the client and server(s) storing the 'kid' and 'kid context' used in the original Observe request, throughout the whole duration of the observation.

Upon leaving the group or before re-joining the group, a group member MUST terminate all the ongoing observations that it has started in the group as observer client.

6.2. Update of Replay Window

Sender Sequence Numbers seen by a server as Partial IV values in request messages can spontaneously increase at a fast pace, for example when a client exchanges unicast messages with other servers using the Group OSCORE Security Context. As in OSCORE [RFC8613], a server always needs to accept such increases and accordingly updates the Replay Window in each of its Recipient Contexts.

As discussed in Section 2.5.1, a newly created Recipient Context would have an invalid Replay Window, if its installation has required to delete another Recipient Context. Hence, the server is not able to verify if a request from the client associated with the new Recipient Context is a replay. When this happens, the server MUST validate the Replay Window of the new Recipient Context, before accepting messages from the associated client (see Section 2.5.1).

Furthermore, when the Group Manager establishes a new Security Context for the group (see Section 2.5.3.2), every server re-initializes the Replay Window in each of its Recipient Contexts.

6.3. Message Freshness

When receiving a request from a client for the first time, the server is not synchronized with the client's Sender Sequence Number, i.e., it is not able to verify if that request is fresh. This applies to a server that has just joined the group, with respect to already present clients, and recurs as new clients are added as group members.

During its operations in the group, the server may also lose synchronization with a client's Sender Sequence Number. This can happen, for instance, if the server has rebooted or has deleted its previously synchronized version of the Recipient Context for that client (see Section 2.5.1).

If the application requires message freshness, e.g., according to time- or event-based policies, the server has to (re-)synchronize with a client's Sender Sequence Number before delivering request messages from that client to the application. To this end, the server can use the approach in Section 10 based on the Echo Option for CoAP [RFC9175], as a variant of the approach defined in Appendix B.1.2 of [RFC8613] applicable to Group OSCORE.

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 Group OSCORE Security Context to process Group OSCORE messages as defined in this document.

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 Signature Algorithm and/or Pairwise Key Agreement 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 conditions holds, a recipient endpoint MUST discard the incoming protected message:

* 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 with an OSCORE group, but the endpoint does not support the pairwise mode.

* The Group Flag is set to 1, and the recipient endpoint retrieves a Security Context which is both valid to process the message and also associated with an OSCORE group, but the endpoint does not support the group mode.

* 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 with an OSCORE group.

  As per Section 6.1 of [RFC8613], this holds also when retrieving a Security Context which is valid but not associated with an OSCORE group. Future specifications may define how to process incoming messages protected with a Security Contexts as in [RFC8613], when the Group Flag bit is set to 1.

Otherwise, if a Security Context associated with 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 with 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 Manager indicates that the group uses (also) the group mode, as part of the group data provided to candidate group members when joining the group.

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 2.5.3.2) 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]. For encryption and decryption operations, the Signature Encryption Algorithm from the Common Context is used.

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 predefined, sufficient amount of time, in order to correctly perform potential 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 when sent over non-multicast transports, 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 reject any message which is malformed and does not follow the format specified in Section 4 of this document, or which is not cryptographically validated in a successful way.

In either case, it is RECOMMENDED that a server does not send back any error message in reply to a received request, if any of the two following conditions holds:

- The server is not able to identify the received request as a group request, i.e., as sent to all servers in the group.

- The server identifies the received request as a group request.

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. The Signature Encryption Algorithm from the Common Context MUST be used.

- In step 5, the countersignature 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 encrypted countersignature (see Section 4.1).

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 2.5.3.1), the client MUST NOT update the stored value associated with 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 3.2), 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 Gid as ID Context (see Section 2.5.3.2), the client MUST NOT update the stored value associated with 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.

* 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 with the OSCORE group, the server MAY respond with a 4.01 (Unauthorized) error message. 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 authentication credential associated with the Recipient ID to be stored in the new Recipient Context. 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.

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

* In step 6, the server also verifies the countersignature, by using the public key from the client's authentication credential stored in 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].

  - The server MUST perform signature verification before decrypting the COSE object, as defined below. Implementations that cannot perform the two steps in this order MUST ensure that no access to the plaintext is possible before a successful signature verification and MUST prevent any possible leak of time-related information that can yield side-channel attacks.

- The server retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

  SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

  where KEYSTREAM is derived as per Section 4.1.1.

  The server verifies the original countersignature SIGNATURE.

- If the signature verification fails, the server SHALL stop processing the request, SHALL NOT update the Replay Window, and MAY respond with a 4.00 (Bad Request) response. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain a string, which, if present, MUST be "Decryption failed" as if the decryption had failed.

- When decrypting the COSE object using the Recipient Key, the Signature Encryption Algorithm from the Common Context MUST be used.

* 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 7.2.1 of [I-D.ietf-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 with 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 2.5.3.1).

* 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 3.2). That is, upon establishing a new Security Context with a new Gid as ID

Context (see Section 2.5.3.2), the server MUST NOT update the stored value associated with 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 3.2).

* 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 3.2), 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. The Signature Encryption Algorithm from the Common Context MUST be used.

  If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request (see Section 3.2), then the new ID Context MUST be included in the 'kid context' parameter of the response.

  The server can obtain a new Sender ID from the Group Manager, when individually rekeyed (see Section 2.5.3.1) or when re-joining the group. In such a case, the server can help the client to synchronize, by including the 'kid' parameter in a response protected in group mode, even when the request was protected in pairwise mode (see Section 9.3).

  That is, when responding to a request protected in pairwise mode, the server SHOULD include the 'kid' parameter in a response protected in group mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

* In step 5, the countersignature 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 encrypted countersignature (see Section 4.1).

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 external_aad structure (see Section 4.3).

- 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 external_aad structure (see Section 4.3).

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 can help the client to synchronize, by including also the 'kid context' parameter in notifications following a group rekeying, with value set to the ID Context (Gid) of the new Security Context.

If there is a known upper limit to the duration of a group rekeying, the server SHOULD include the 'kid context' parameter during that time. Otherwise, the server SHOULD include it until the Max-Age has expired for the last notification sent before the installation of the new Security Context.

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 request, and that, upon the establishment of a new Security Context,

the client re-initializes its Replay Windows in its Recipient Contexts (see Section 3.2).

* In step 2, the decoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, a 'kid' may not be present, if the response is a reply to a request protected in pairwise mode. In such a case, the client assumes the response 'kid' to be the Recipient ID for the server to which the request protected in pairwise mode was intended for.

  If the response 'kid context' matches an existing ID Context (Gid) but the received/assumed '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 authentication credential associated with the Recipient ID to be stored in the new Recipient Context. If the application does not specify dynamic derivation of new Recipient Contexts, then the client SHALL stop processing the response.

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

* In step 5, the client also verifies the countersignature, by using the public key from the server's authentication credential stored in the associated Recipient Context. In particular:

  - The client MUST perform signature verification before decrypting the COSE object, as defined below. Implementations that cannot perform the two steps in this order MUST ensure that no access to the plaintext is possible before a successful signature verification and MUST prevent any possible leak of time-related information that can yield side-channel attacks.

  - The client retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

    SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

    where KEYSTREAM is derived as per Section 4.1.1.

    The client verifies the original countersignature SIGNATURE.

  - If the verification of the countersignature fails, the server SHALL stop processing the response, and SHALL NOT update the Notification Number associated with the server if the response is an Observe notification [RFC7641].

- After a successful verification of the countersignature, the client performs also the following actions if the response is not an Observe notification.

  - In case the request was protected in pairwise mode and the 'kid' parameter is present in the response, the client checks whether this received 'kid' is equal to the expected 'kid', i.e., the known Recipient ID for the server to which the request was intended for.

  - In case the request was protected in pairwise mode and the 'kid' parameter is not present in the response, the client checks whether the server that has sent the response is the same one to which the request was intended for. This can be done by checking that the public key used to verify the countersignature of the response is equal to the public key included in the authentication credential Recipient Auth Cred, which was taken as input to derive the Pairwise Sender Key used for protecting the request (see Section 2.4.1).

  In either case, if the client determines that the response has come from a different server than the expected one, then the client SHALL discard the response and SHALL NOT deliver it to the application. Otherwise, the client hereafter considers the received 'kid' as the current Recipient ID for the server.

- When decrypting the COSE object using the Recipient Key, the Signature Encryption Algorithm from the Common Context MUST be used.

* 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.

* 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 external_aad structure (see Section 4.3).

* 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 external_aad structure (see Section 4.3).

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 2.5.3.1), as well as when it installs a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 3.2).

* The ordering and the replay protection of notifications received from a server are performed as per Sections 4.1.3.5.2 and 7.4.1 of [RFC8613], by using the Notification Number associated with that server for the observation in question. In addition, the client performs the following actions for each ongoing observation.

  - When receiving the first valid notification from a server, the client MUST store the current kid "kid1" of that server for the observation in question. If the 'kid' field is included in the OSCORE option of the notification, its value specifies "kid1". If the Observe request was protected in pairwise mode (see Section 9.3), the 'kid' field may not be present in the OSCORE option of the notification (see Section 4.2). In this case, the client assumes "kid1" to be the Recipient ID for the server to which the Observe request was intended for.

  - When receiving another valid notification from the same server - which can be identified and recognized through the same public key used to verify the countersignature and included in the server's authentication credential - the client determines the current kid "kid2" of the server as above for "kid1", and MUST check whether "kid2" is equal to the stored "kid1". If "kid1" and "kid2" are different, the client MUST cancel or re-register the observation in question.

    Note that, if "kid2" is different from "kid1" and the 'kid' field is omitted from the notification - which is possible if the Observe request was protected in pairwise mode - then the client will compute a wrong keystream to decrypt the countersignature (i.e., by using "kid1" rather than "kid2" in the 'id' field of the 'info' array in Section 4.1.1), thus subsequently failing to verify the countersignature and discarding the notification.

This ensures that the client remains able to correctly perform the ordering and replay protection of notifications, even in case a server legitimately starts using a new Sender ID, as received from the Group Manager when individually rekeyed (see Section 2.5.3.1) or when re-joining the group.

8.5. External Signature Checkers

When receiving a message protected in group mode, a signature checker (see Section 3.1) proceeds as follows.

* The signature checker retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

  SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

  where KEYSTREAM is derived as per Section 4.1.1.

* The signature checker verifies the original countersignature SIGNATURE, by using the public key of the sender endpoint as included in that endpoint's authentication credential. The signature checker determines the right authentication credential based on the ID Context (Gid) and the Sender ID of the sender endpoint.

Note that the following applies when attempting to verify the countersignature of a response message.

* The response may not include a Partial IV and/or an ID Context. In such a case, the signature checker considers the same values from the corresponding request, i.e., the request matching with the response by CoAP Token value.

* The response may not include a Sender ID. This can happen when the response protected in group mode matches a request protected in pairwise mode (see Section 9.1), with a case in point provided by [I-D.amsuess-core-cachable-oscore]. In such a case, the signature checker needs to use other means (e.g., source addressing information of the server endpoint) to identify the correct authentication credential including the public key to use for verifying the countersignature of the response.

The particular actions following a successful or unsuccessful verification of the countersignature are application specific and out of the scope of this document.

9. Message Processing in Pairwise Mode

When using the pairwise mode of Group OSCORE, messages are protected and processed as in [RFC8613], 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 in a group that uses also the group 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. For encryption and decryption operations, the AEAD Algorithm from the Common Context is used (see Section 2.1.1).

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).

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 [RFC9175] 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 3.8 of [I-D.ietf-core-groupcomm-bis]). This prevents the attack described in Section 12.9, which leverages requests sent over unicast to a single group member and protected with the group mode.

Senders cannot use the pairwise mode to protect a message intended for multiple recipients. In fact, the pairwise mode is defined only between two endpoints and the keying material is thus only available to one recipient.

However, a sender can use the pairwise mode to protect a message sent to (but not intended for) multiple recipients, if interested in a response from only one of them. For instance, this is useful to support the address discovery service defined in Section 9.1, when a single 'kid' value is indicated in the payload of a request sent to multiple recipients, e.g., over multicast.

The Group Manager indicates 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 authentication credential and the Recipient ID for the recipient endpoint, as stored in the Recipient Context associated with that endpoint (see Section 2.4.4).

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 authentication credential 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 set of servers, identified by their 'kid' values specified in the request payload. 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. Main Differences from OSCORE

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 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 2.4 of this document.

9.3. Protecting the Request

When using the pairwise mode, the request is protected as defined in Section 8.1 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following difference also applies.

* If Observe [RFC7641] is supported, what is defined in Section 8.1.1 of this document holds.

9.4. 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 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

* If the server discards the request due to not retrieving a Security Context associated with the OSCORE group or to not supporting the pairwise mode, the server MAY respond with a 4.01 (Unauthorized) error message or a 4.02 (Bad Option) error message, respectively. 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 a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.4.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.

* If Observe [RFC7641] is supported, what is defined in Section 8.2.1 of this document holds.

9.5. Protecting the Response

When using the pairwise mode, a response is protected as defined in Section 8.3 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

* If the server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.2), 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.

* If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request (see Section 3.2), then the new ID Context MUST be included in the 'kid context' parameter of the response.

* The server can obtain a new Sender ID from the Group Manager, when individually rekeyed (see Section 2.5.3.1) or when re-joining the group. In such a case, the server can help the client to synchronize, by including the 'kid' parameter in a response protected in pairwise mode, even when the request was also protected in pairwise mode.

  That is, when responding to a request protected in pairwise mode, the server SHOULD include the 'kid' parameter in a response protected in pairwise mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

* If Observe [RFC7641] is supported, what is defined in Section 8.3.1 of this document holds.

9.6. 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 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

* The client may receive a response protected with a Security Context different from the one used to protect the corresponding request. Also, upon the establishment of a new Security Context, the client re-initializes its Replay Windows in its Recipient Contexts (see Section 2.2).

* The same as described in Section 8.4 holds with respect to handling the 'kid' parameter of the response, when received as a reply to a request protected in pairwise mode. The client can also in this case check whether the replying server is the expected one, by relying on the server's public key. However, since the response is protected in pairwise mode, the public key is not used for verifying a countersignature as in Section 8.4. Instead, the expected server's authentication credential - namely Recipient Auth Cred and including the server's public key - was taken as input to derive the Pairwise Recipient Key used to decrypt and verify the response (see Section 2.4.1).

* If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.4.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.

* If Observe [RFC7641] is supported, what is defined in Section 8.4.1 of this document holds. The client can also in this case identify a server to be the same one across a change of Sender ID, by relying on the server's public key. As to the expected server's authentication credential, the same holds as specified above for non-notification responses.

10. Challenge-Response Synchronization

This section describes how a server endpoint can synchronize with Sender Sequence Numbers of client endpoints in the group. Similarly to what is defined in Appendix B.1.2 of [RFC8613], the server performs a challenge-response exchange with a client, by using the Echo Option for CoAP specified in Section 2 of [RFC9175].

Upon receiving a request from a particular client for the first time, the server processes the message as described in this document, 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 Echo option value SHOULD NOT be reused; when it is reused, it MUST be highly

unlikely to have been recently used with this client. 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 in the response 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. These are specified in the 'kid context' and 'kid' fields of the OSCORE Option of the 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 3.2), the server MUST delete all the stored Echo values associated with members of the group.

Upon receiving a 4.01 (Unauthorized) response that includes an Echo Option and originates from a verified group member, the 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.

If the group uses also the group mode and the used Signature Algorithm supports ECDH (e.g., ECDSA, EdDSA), the client MUST use the pairwise mode to protect the request, as per Section 9.3. Note that, as defined in Section 9, endpoints that are members of such a group and that use the Echo Option MUST support the pairwise mode.

The client does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This allows 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 preconfigured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed, 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.

* 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.

* If the server stores an Echo Option value for the pair (gid,kid) associated with 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, hence performing a new challenge-response exchange.

If the verifications above are successful, the server proceeds as follows. In case the Replay Window in the Recipient Context associated with the client has not been set yet, the server updates the Replay Window to mark the current Sender Sequence Number from the latest received request as seen (but all newer ones as new), and delivers the message as fresh to the application. Otherwise, the server discards the verification result and treats the message as fresh or as a replay, according to the existing Replay Window.

A server should not deliver 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. A server may perform the challenge-response described above at any time, if synchronization with Sender Sequence Numbers of clients is lost, e.g., after a device reboot. A client has to be ready to perform the challenge-response based on the Echo Option if 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, 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.

In order to ameliorate this, the server may 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, preconfigured 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 section 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.

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. Thus, 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 victim of the attack discussed in Section 12.9, in case such requests are protected with the group mode. Instead, protecting those requests with the pairwise mode prevents the attack above. In fact, only the exact server involved in the challenge-response exchange is able to derive the 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, even if the group mode was used, this would require the adversary to forge the countersignature of both 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.

11. Implementation Compliance

Like in [RFC8613], HKDF SHA-256 is the mandatory to implement HKDF.

An endpoint may support only the group mode, or only the pairwise mode, or both.

For endpoints that support the group mode, the following applies.

* For endpoints that use authenticated encryption, the AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as Signature Encryption Algorithm (see Section 2.1.4).

* For many constrained IoT devices it is problematic to support more than one signature algorithm. Existing devices can be expected to support either EdDSA or ECDSA. In order to enable as

much interoperability as we can reasonably achieve, the following applies with respect to the Signature Algorithm (see Section 2.1.5).

Less constrained endpoints SHOULD implement both: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032]; and the ECDSA signature algorithm together with the elliptic curve P-256.

Constrained endpoints SHOULD implement: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032]; or the ECDSA signature algorithm together with the elliptic curve P-256.

* Endpoints that implement the ECDSA signature algorithm MAY use "deterministic ECDSA" as specified in [RFC6979]. Pure deterministic elliptic-curve signature algorithms such as deterministic ECDSA and EdDSA have the advantage of not requiring access to a source of high-quality randomness. However, these signature algorithms have been shown vulnerable to some side-channel and fault injection attacks due to their determinism, which can result in extracting a device's private key. As suggested in Section 2.1.1 of [I-D.ietf-cose-rfc8152bis-algs], this can be addressed by combining both randomness and determinism [I-D.mattsson-cfrg-det-sigs-with-noise].

For endpoints that support the pairwise mode, the following applies.

* The AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as AEAD Algorithm (see Section 2.1.1).

* The ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as Pairwise Key Agreement Algorithm (see Section 2.1.7).

* In order to enable as much interoperability as we can reasonably achieve in the presence of constrained devices (see above), the following applies.

  Less constrained endpoints SHOULD implement both the X25519 curve [RFC7748] and the P-256 curve as ECDH curves.

  Constrained endpoints SHOULD implement the X25519 curve [RFC7748] or the P-256 curve as ECDH curve.

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].

12. 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 countersignatures, as discussed in Section 12.1.

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.

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 12.3 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 differences. First, the 'kid context' of request messages is part of the Additional Authenticated Data, thus safely enabling to keep observations active beyond a possible change of ID Context (Gid), following a group rekeying (see Section 4.3). Second, the countersignature included in a Group OSCORE message protected in group mode is computed also over the value of the OSCORE option, which is also part of the Additional Authenticated Data used in the signing process. This is further discussed in Section 12.7 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.

12.1. Security of the Group Mode

The group mode defined in Section 8 relies on commonly shared group keying material to protect communication within a group. Using the group mode has the implications discussed below. The following refers to group members as the endpoints in the group storing the latest version of the group keying material.

* 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.

* If the used encryption algorithm provides integrity protection, then it also ensures group authentication and proof of group membership, but not source authentication. That is, 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. In fact, any group member is able to derive the Sender Key used by the actual sender endpoint, and thus can compute a valid authentication tag. Therefore, the message content could originate from any of the current group members.

  Furthermore, if the used encryption algorithm does not provide integrity protection, then it does not ensure any level of message authentication or proof of group membership.

  On the other hand, proof of group membership is always ensured by construction through the strict management of the group keying material (see Section 3.2). That is, the group is rekeyed in case of members' leaving, and the current group members are informed of former group members. Thus, a current group member storing the latest group keying material does not store the authentication credential of any former group member.

  This allows a recipient endpoint to rely on the stored authentication credentials and public keys included therin, in order to always confidently assert the group membership of a sender endpoint when processing an incoming message, i.e., to assert that the sender endpoint was a group member when it signed the message. In turn, this prevents a former group member to possibly re-sign and inject in the group a stored message that was protected with old keying material.

* Source authentication of messages sent to a group is ensured through a countersignature, which is computed by the sender using its own private key and then appended to the message payload.

Also, the countersignature is encrypted by using a keystream derived from the group keying material (see Section 4.1). This ensures group privacy, i.e., an attacker cannot track an endpoint over two groups by linking messages between the two groups, unless being also a member of those groups.

The security properties of the group mode are summarized below.

1. Asymmetric source authentication, by means of a countersignature.

2. Symmetric group authentication, by means of an authentication tag (only for encryption algorithms providing integrity protection).

3. Symmetric group confidentiality, by means of symmetric encryption.

4. Proof of group membership, by strictly managing the group keying material, as well as by means of integrity tags when using an encryption algorithm that provides also integrity protection.

5. Group privacy, by encrypting the countersignature.

The group mode fulfills the security properties above while also displaying the following benefits. First, the use of an encryption algorithm that does not provide integrity protection results in a minimal communication overhead, by limiting the message payload to the ciphertext and the encrypted countersignature. Second, it is possible to deploy semi-trusted entities such as signature checkers (see Section 3.1), which can break property 5, but cannot break properties 1, 2 and 3.

12.2. Security of the Pairwise Mode

The pairwise mode defined 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 2.4).

The used encryption algorithm MUST provide integrity protection. Therefore, the pairwise mode ensures both pairwise data-confidentiality and source authentication of messages, without using countersignatures. Furthermore, the recipient endpoint achieves proof of group membership for the sender endpoint, since only current group members have the required keying material to derive a valid Pairwise Sender/Recipient Key.

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.

12.3. 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:

- Uniqueness of Sender IDs within the group is enforced by the Group Manager. In fact, from the moment when a Gid is assigned to a group until the moment a new Gid is assigned to that same group, the Group Manager does not reassign a Sender ID within the group (see Section 3.2).

- 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.

- 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.

12.4. Management of Group Keying Material

The approach described in this document 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 MUST be adopted.

[I-D.ietf-ace-key-groupcomm-oscore] specifies 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 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.

12.5. 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 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 request.

In particular, a server may first get a 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 10.3 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.

12.5.1. 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 a 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 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.

12.5.2. 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.

12.6. 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.

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

12.7. Cross-group Message Injection

A same endpoint is allowed to and would likely use the same pair (private key, authentication credential) 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 12.7.1).

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 12.7.2, the attack is practically infeasible if the message is protected in group mode, thanks to the countersignature also bound to the OSCORE option through the Additional Authenticated Data used in the signing process (see Section 4.3).

12.7.1. 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 pair (private key, authentication credential) 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 authentication credential of X in G1 and G2, respectively.

- A recipient endpoint Y which is member of both G1 and G2, and uses the same pair (private key, authentication credential) 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 authentication credential 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 Sender Keys used by 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 attempt to 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 set to G2 (rather than G1); and the 'kid' is set to Sid2 (rather than Sid1). 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 the Pairwise Recipient Key associated with 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.

12.7.2. Attack Prevention in Group Mode

When a Group OSCORE message is protected with the group mode, the countersignature 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).

That is, other than over the ciphertext, the countersignature is computed over: the ID Context (Gid) 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 group requests as well as in responses to requests protected in group mode.

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 12.7.1, since it would require the adversary to additionally forge a valid countersignature that replaces the original one in the forged message M2.

If, hypothetically, the countersignature did not cover the OSCORE option:

* The attack described in Section 12.7.1 would still be possible against response messages protected in group mode, since the same unchanged countersignature from message M1 would be also valid in message M2.

*A simplification would also be possible in performing the attack, since Z is able to derive the Sender/Recipient Keys of X and Y in G1 and G2. That is, Z can also set a convenient Partial IV in the response, until the same unchanged MAC is successfully verified by using G2 as 'request_kid_context', Sid2 as 'request_kid', and the symmetric key associated with 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. 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 the same group G1.

12.8. Prevention of Group Cloning Attack

Both when using the group mode and the pairwise mode, the message protection covers also the Group Manager's authentication credential. This is included in the Additional Authenticated Data used in the signing process and/or in the integrity-protected encryption process (see Section 4.3).

By doing so, an endpoint X member of a group G1 cannot perform the following attack.

1. X sets up a group G2 where it acts as Group Manager.

2. X makes G2 a "clone" of G1, i.e., G1 and G2 use the same algorithms and have the same Master Secret, Master Salt and ID Context.

3. X collects a message M sent to G1 and injects it in G2.

4. Members of G2 accept M and believe it to be originated in G2.

The attack above is effectively prevented, since message M is protected by including the authentication credential of G1's Group Manager in the Additional Authenticated Data. Therefore, members of G2 do not successfully verify and decrypt M, since they correctly use the authentication credential of X as Group Manager of G2 when attempting to.

12.9. 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 9.3, in order to protect a request sent to a single group member by using pairwise keying material (see Section 2.4). 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 7.2.1 of [I-D.ietf-core-observe-multicast-notifications].

With particular reference to block-wise transfers [RFC7959], Section 3.8 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 Section 10, with respect to requests including a CoAP Echo Option [RFC9175] that have to be sent over unicast, as a challenge-response method for servers to achieve synchronization of clients' Sender Sequence Number.

12.10. 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.

12.11. 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].

12.12. Replay Protection

As in OSCORE [RFC8613], 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 2.5.3, 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 12.3) is preserved also when a new Security Context is established.

Since one-to-many communication such as multicast usually involves unreliable transports, the simplification of the Replay Window to a size of 1 suggested in Section 7.4 of [RFC8613] is not viable with Group OSCORE, unless exchanges in the group rely only on unicast messages.

As discussed in Section 6.2, a Replay Window may be initialized as not valid, following the loss of mutable Security Context Section 2.5.1. In particular, Section 2.5.1.1 and Section 2.5.1.2 define measures that endpoints need to take in such a situation, before resuming to accept incoming messages from other group members.

12.13. Message Freshness

As discussed in Section 6.3, a server may not be able to assert whether an incoming request is fresh, in case it does not have or has lost synchronization with the client's Sender Sequence Number.

If freshness is relevant for the application, the server may (re-)synchronize with the client's Sender Sequence Number at any time, by using the approach described in Section 10 and based on the CoAP Echo Option [RFC9175], as a variant of the approach defined in Appendix B.1.2 of [RFC8613] applicable to Group OSCORE.

12.14. Client Aliveness

Building on Section 12.5 of [RFC8613], a server may use the CoAP Echo Option [RFC9175] to verify the aliveness of the client that originated a received request, by using the approach described in Section 10 of this document.

12.15. 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 2.5.2, 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 group mode uses the "encrypt-then-sign" construction, i.e., the countersignature is computed over the COSE_Encrypt0 object (see Section 4.1). This is motivated by enabling additional entities acting as signature checkers (see Section 3.1), which do not join a group as members but are allowed to verify countersignatures of messages protected in group mode without being able to decrypt them (see Section 8.5).

If the encryption algorithm used in group mode provides integrity protection, countersignatures of COSE_Encrypt0 with short authentication tags do not provide the security properties associated with the same algorithm used in COSE_Sign (see Section 6 of [I-D.ietf-cose-countersign]). To provide 128-bit security against collision attacks, the tag length MUST be at least 256-bits. A countersignature of a COSE_Encrypt0 with AES-CCM-16-64-128 provides at most 32 bits of integrity protection.

The derivation of pairwise keys defined in Section 2.4.1 is compatible with ECDSA and EdDSA asymmetric keys, but is not compatible with RSA asymmetric keys.

For the public key translation from Ed25519 (Ed448) to X25519 (X448) specified in Section 2.4.1, variable time methods can be used since the translation operates on public information. Any byte string of appropriate length is accepted as a public key for X25519 (X448) in [RFC7748]. It is therefore not necessary for security to validate the translated public key (assuming the translation was successful).

The security of using the same key pair for Diffie-Hellman and for signing (by considering the ECDH procedure in Section 2.4 as a Key Encapsulation Mechanism (KEM)) is demonstrated in [Degabriele] and [Thormarker].

Applications using ECDH (except X25519 and X448) based KEM in Section 2.4 are assumed to verify that a peer endpoint's public key is on the expected curve and that the shared secret is not the point at infinity. The KEM in [Degabriele] checks that the shared secret is different from the point at infinity, as does the procedure in Section 5.7.1.2 of [NIST-800-56A] which is referenced in Section 2.4.

Extending Theorem 2 of [Degabriele], [Thormarker] shows that the same key pair can be used with X25519 and Ed25519 (X448 and Ed448) for the KEM specified in Section 2.4. By symmetry in the KEM used in this document, both endpoints can consider themselves to have the recipient role in the KEM - as discussed in Section 7 of [Thormarker] - and rely on the mentioned proofs for the security of their key pairs.

Theorem 3 in [Degabriele] shows that the same key pair can be used for an ECDH based KEM and ECDSA. The KEM uses a different KDF than in Section 2.4, but the proof only depends on that the KDF has certain required properties, which are the typical assumptions about HKDF, e.g., that output keys are pseudorandom. In order to comply with the assumptions of Theorem 3, received public keys MUST be successfully validated, see Section 5.6.2.3.4 of [NIST-800-56A]. The validation MAY be performed by a trusted Group Manager. For [Degabriele] to apply as it is written, public keys need to be in the expected subgroup. For this we rely on cofactor DH, Section 5.7.1.2 of [NIST-800-56A] which is referenced in Section 2.4.

HashEdDSA variants of Ed25519 and Ed448 are not used by COSE, see Section 2.2 of [I-D.ietf-cose-rfc8152bis-algs], and are not covered by the analysis in [Thormarker]. Hence, they MUST NOT be used with the public keys used to derive pairwise keys as specified in this document.

12.16. Message Segmentation

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

12.17. 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, with the following addition.

* When protecting a message in group mode, the countersignature is encrypted by using a keystream derived from the group keying material (see Section 4.1 and Section 4.1.1). This ensures group privacy. That is, an attacker cannot track an endpoint over two groups by linking messages between the two groups, unless being also a member of those groups.

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

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

- 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 approach described in Section 10 to achieve Sender 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 approach described in Section 12.6 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.

13. IANA Considerations

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

This document has the following actions for IANA.

13.1. OSCORE Flag Bits Registry

IANA is asked to add the following value entry to the "OSCORE Flag Bits" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Bit Position	Name	Description	Reference
2	Group Flag	For using a Group OSCORE Security Context, set to 1 if the message is protected with the group mode	[This Document]

14. References

14.1. Normative References

[I-D.ietf-core-groupcomm-bis] Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-06, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>.

[I-D.ietf-cose-countersign] Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress, Internet-Draft, draft-ietf-cose-countersign-05, 23 June 2021, <https://www.ietf.org/archive/id/draft-ietf-cose-countersign-05.txt>.

[I-D.ietf-cose-rfc8152bis-algs] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>.

[I-D.ietf-cose-rfc8152bis-struct] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>.

[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>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[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>.

[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>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[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>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/info/rfc9175>.

14.2. Informative References

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

[I-D.amsuess-core-cachable-oscore] Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-04, 6 March 2022, <https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-04.txt>.

[I-D.ietf-ace-key-groupcomm] Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-15, 23 December 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-15.txt>.

[I-D.ietf-ace-key-groupcomm-oscore] Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>.

[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)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>.

[I-D.ietf-core-observe-multicast-notifications] Tiloca, M., Höglund, R., Amsüss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work

in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-03, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-03.txt>.

[I-D.ietf-cose-cbor-encoded-cert]
Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-03, 10 January 2022, <https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.txt>.

[I-D.ietf-lwig-curve-representations]
Struik, R., "Alternative Elliptic Curve Representations", Work in Progress, Internet-Draft, draft-ietf-lwig-curve-representations-23, 21 January 2022, <https://www.ietf.org/archive/id/draft-ietf-lwig-curve-representations-23.txt>.

[I-D.ietf-lwig-security-protocol-comparison] Mattsson, J. P., Palombini, F., and M. Vucinic, "Comparison of CoAP Security Protocols", Work in Progress, Internet-Draft, draft-ietf-lwig-security-protocol-comparison-05, 2 November 2020, <https://www.ietf.org/archive/id/draft-ietf-lwig-security-protocol-comparison-05.txt>.

[I-D.ietf-tls-dtls13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>.

[I-D.mattsson-cfrg-det-sigs-with-noise] Mattsson, J. P., Thormarker, E., and S. Ruohomaa, "Deterministic ECDSA and EdDSA Signatures with Additional Randomness", Work in Progress, Internet-Draft, draft-mattsson-cfrg-det-sigs-with-noise-04, 15 February 2022, <https://www.ietf.org/archive/id/draft-mattsson-cfrg-det-sigs-with-noise-04.txt>.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[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>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[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>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[Thormarker] Thormarker, E., "On using the same key pair for Ed25519 and an X25519 based KEM", April 2021, <https://eprint.iacr.org/2021/509>.

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 document. There can be a one-to-one or a one-to-many relation between security groups and application groups, and vice versa.

* CoAP group, i.e., a set of CoAP endpoints where each endpoint is configured to receive one-to-many CoAP requests, e.g., sent to the group's associated IP multicast address and UDP port as defined in [I-D.ietf-core-groupcomm-bis]. 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 points 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. Also, every client expects and is able to handle multiple response messages associated with 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. One should not assume that the set of members of a security group remains fixed. That is, the group membership is subject to changes, possibly on a frequent basis.

*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 use the same 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 lighting 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.

* Integrated building control: enabling Building Automation and Control Systems (BACSs) to control multiple heating, ventilation and air-conditioning units to predefined 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.

*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.

*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.

*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.

*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-tolerant 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 with 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 3.2).

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, i.e., before the Group Manager terminates the group or starts reassigning Gid values to the group (see Section 3.2). 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 12.6, 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 favorable 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, including its own authentication credential (see Section 2). The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

As mentioned in Section 3, the Group Manager and the join process can be as specified in [I-D.ietf-ace-key-groupcomm-oscore].

Appendix E. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

E.1. Version -13 to -14

*Replaced "node" with "endpoint" where appropriate.

*Replaced "owning" with "storing" (of keying material).

*Distinction between "authentication credential" and "public key".

*Considerations on storing whole authentication credentials.

*Considerations on Denial of Service.

* Recycling of Group IDs by tracking the "Birth Gid" of each group member is now optional to support and use for the Group Manager.

* Fine-grained suppression of error responses.

* Changed section title "Mandatory-to-Implement Compliance Requirements" to "Implementation Compliance".

* "Challenge-Response Synchronization" moved to the document body.

* RFC 7641 and draft-ietf-core-echo-request-tag as normative references.

* Clarifications and editorial improvements.

E.2. Version -12 to -13

* Fixes in the derivation of the Group Encryption Key.

* Added Mandatory-to-Implement compliance requirements.

* Changed UCCS to CCS.

E.3. Version -11 to -12

* No mode of operation is mandatory to support.

* Revised parameters of the Security Context, COSE object and external_aad.

* Revised management of keying material for the Group Manager.

* Informing of former members when rekeying the group.

* Admit encryption-only algorithms in group mode.

* Encrypted countersignature through a keystream.

* Added public key of the Group Manager as key material and protected data.

* Clarifications about message processing, especially notifications.

* Guidance for message processing of external signature checkers.

* Updated derivation of pairwise keys, with more security considerations.

* Termination of ongoing observations as client, upon leaving or before re-joining the group.

* Recycling Group IDs by tracking the "Birth Gid" of each group member.

* Expanded security and privacy considerations about the group mode.

* Removed appendices on skipping signature verification and on COSE capabilities.

* Fixes and editorial improvements.

E.4. Version -10 to -11

* Loss of Recipient Contexts due to their overflow.

* Added diagram on keying material components and their relation.

* Distinction between anti-replay and freshness.

* Preservation of Sender IDs over rekeying.

* Clearer cause-effect about reset of SSN.

* The GM provides public keys of group members with associated Sender IDs.

* Removed 'par_countersign_key' from the external_aad.

* One single format for the external_aad, both for encryption and signing.

* Presence of 'kid' in responses to requests protected with the pairwise mode.

* Inclusion of 'kid_context' in notifications following a group rekeying.

* Pairwise mode presented with OSCORE as baseline.

* Revised examples with signature values.

* Decoupled growth of clients' Sender Sequence Numbers and loss of synchronization for server.

* Sender IDs not recycled in the group under the same Gid.

* Processing and description of the Group Flag bit in the OSCORE option.

* Usage of the pairwise mode for multicast requests.

*Clarifications on synchronization using the Echo option.

*General format of context parameters and external_aad elements, supporting future registered COSE algorithms (new Appendix).

*Fixes and editorial improvements.

E.5. Version -09 to -10

*Removed 'Counter Signature Key Parameters' from the Common Context.

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

*New countersignature 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.

E.6. 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.

E.7. 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.

E.8. 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.

E.9. 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.

E.10. 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).

E.11. 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 countersignature 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 identifier 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 countersignature for outgoing messages (see Sections 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).

E.12. 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.

E.13. 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.

E.14. 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 establishment/derivation of Security Contexts.

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

*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.

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

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

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

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

*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, Martin Gunnarsson, Klaus Hartke, Rikard Hoeglund, Richard Kelsey, 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
SE-16440 Stockholm Kista
Sweden

Email: marco.tiloca@ri.se

Göran Selander
Ericsson AB
Torshamnsgatan 23

SE-16440 Stockholm Kista
Sweden

Email: goran.selander@ericsson.com

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

Email: francesca.palombini@ericsson.com

John Preuss Mattsson
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden

Email: john.mattsson@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
45127 Essen
Germany

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