CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

C. Amsüss

M. Tiloca
RISE AB

2 March 2026

End-to-End Protected and Cacheable Responses for the Constrained Application Protocol (CoAP) using Group Object Security for Constrained RESTful Environments (Group OSCORE)
draft-ietf-core-cacheable-oscore-01

Abstract

When using the Constrained Application Protocol (CoAP), exchanged messages can be protected end-to-end also across untrusted intermediary proxies. This can be achieved with Object Security for Constrained RESTful Environments (OSCORE) or, in the case of group communication, with Group Object Security for Constrained RESTful Environments (Group OSCORE). However, this sidesteps the proxies' abilities to cache responses from the origin server(s). This document restores cacheability of end-end protected responses at proxies, by using Group OSCORE and introducing consensus requests, which any client in an OSCORE group can send to one server or multiple servers in the same group.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/.

Source for this draft and an issue tracker can be found at https://github.com/core-wg/cacheable-oscore.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports intermediary proxies that perform requests on behalf of clients and relay back responses. A core functionality of intermediary proxies is the caching of responses.

Even in the presence of such intermediaries, exchanged CoAP messages can be protected end-to-end by using Object Security for Constrained RESTful Environments (OSCORE) [RFC8613].

In a group communication environment [I-D.ietf-core-groupcomm-bis], CoAP exchanged messages can be protected end-to-end by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm]. When protected with the group mode of Group OSCORE (see Section 7 of [I-D.ietf-core-oscore-groupcomm]), requests and responses exchanged in the OSCORE group can be read by all group members, i.e., not only by the intended recipient(s), thus achieving group-level confidentiality.

With any security mechanism for CoAP, the caching of responses at intermediaries presents operators with a trade-off between required trust and provided performance:

* If an intermediary proxy is trusted, the proxy can inspect traffic that is going through, cache it, and provide cached responses.

* An untrusted proxy, while possible with OSCORE and Group OSCORE, sees different ciphertexts for different requests even when those requests target the same resource. Even if the proxy could somehow produce a cache hit, cached responses would be rejected by the request-response matching of (Group) OSCORE.

By using Group OSCORE, this document provides a way out of the trade-off situation.

In particular, this document enables cacheability of protected responses for proxies that are not members of the OSCORE group and that are unaware of OSCORE and Group OSCORE in general. To this end, it builds on the concept of "consensus request" initially considered in [I-D.ietf-core-observe-multicast-notifications], and it defines "Deterministic Request" as a convenient incarnation of such concept.

All clients wishing to send a particular request with the GET method or FETCH method [RFC8132] are able to deterministically compute the same protected request, using a variation of the pairwise mode of Group OSCORE (see Section 8 of [I-D.ietf-core-oscore-groupcomm]). It follows that cache hits become possible at the proxy, which can thus serve clients in the group from its cache. Like in [I-D.ietf-core-observe-multicast-notifications], this requires that clients and servers are already members of a suitable OSCORE group.

Cacheability of protected responses is useful also in applications where several clients wish to retrieve the same object from a single server. Some security properties of OSCORE are dispensed with, in order to gain other desirable properties.

In order to clearly handle the protocol’s security properties and to broaden applicability to group situations outside the deterministic case, the technical implementation is split into two halves:

* Maintaining request-response bindings in the absence of request source authentication; and

* Building and processing of Deterministic Requests, which have no source authentication and thus require the former.

1.1. Use Cases

When firmware updates are delivered using CoAP, many similar devices fetch the same large data at the same time. Collecting such large data at a proxy from its cache not only keeps the traffic low, but also lets the clients ride single file to hide their numbers [SW-EPIV] and identities. By using protected Deterministic Requests as defined in this document, it is possible to efficiently perform

data collection at a proxy also when the firmware updates are protected end-to-end.

When relying on intermediaries to fan out the delivery of multicast data protected end-to-end as in [I-D.ietf-core-multicast-notifications-proxy], the use of protected Deterministic Requests as defined in this document allows for a more efficient setup, by reducing the amount of message exchanges and enabling early population of cache entries (see Appendix C).

When building RESTful networks following the patterns of Information-Centric Networking (ICN), CoAP proxies take the role of forwarding nodes. Caching plays a large role in such networks, and cacheable OSCORE provides a compatible security mechanism [ICN-paper].

When DNS messages are transported over CoAP [I-D.ietf-core-dns-over-coap], it is recommended to use OSCORE for protecting such messages [DNS-CoAP-paper]. By restoring cacheability of OSCORE-protected responses, it becomes possible to benefit from the cache retrieval of such CoAP responses that particularly transport DNS messages.

1.2. 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 terms and concepts of CoAP [RFC7252] and its method FETCH [RFC8132], group communication for CoAP [I-D.ietf-core-groupcomm-bis], Concise Binary Object Representation (CBOR) [RFC8949], CBOR Object Signing and Encryption (COSE) [RFC9052][RFC9053], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This document also introduces the following new terms.

* Consensus Request: a CoAP request that multiple clients use to repeatedly access a particular resource. In this document, it exclusively refers to requests protected with Group OSCORE to a resource hosted at one or more servers in the OSCORE group.

A Consensus Request has all the properties relevant to caching, but its transport dependent properties (e.g., Token or Message ID) are not defined. Thus, different requests on the wire can be said to "be the same Consensus Request" even if they have different Tokens or source addresses.

The Consensus Request is the reference for request-response binding. In general, a client processing a response to a Consensus Request did not generate (and thus sign) the consensus request. The client not only needs to decrypt the Consensus Request to understand a corresponding response (for example to tell which path was requested), but it also needs to verify that this is the only Consensus Request that could elicit this response.

* Deterministic Client: a fictitious member of an OSCORE group, having no Sender Sequence Number, no asymmetric key pair, and no Recipient Context.

  The Group Manager responsible for the OSCORE group (see Section 12 of [I-D.ietf-core-oscore-groupcomm]) sets up the Deterministic Client and assigns it a unique Sender ID like for other group members. Furthermore, the Deterministic Client has only the minimum common set of privileges shared by all group members.

* Deterministic Request: a Consensus Request generated by the Deterministic Client. The use of Deterministic Requests is defined in Section 3.

* Ticket Request: a Consensus Request generated by the server itself.

  This term is not used in the main document, but is useful in comparison with other applications of Consensus Requests that are generated in a different way than as Deterministic Requests. The prototypical Ticket Request is the Phantom Request defined in [I-D.ietf-core-observe-multicast-notifications].

  In Appendix B, the term is used to bridge the gap with that document.

2. OSCORE Message Processing without Source Authentication

In OSCORE [RFC8613], a response is cryptographically bound to the corresponding request through CBOR data items [RFC8949] in their authenticated encryption's Additional Authenticated Data (AAD): "request_kid" and "request_piv". Group OSCORE adds "request_kid_context" to that list. Hereafter, those items are referred to as "request_details".

The security of such binding depends on the server obtaining source authentication for the request, and on that source matching the request_details. If this precondition is not fulfilled, a malicious group member could alter a request to the server (without altering the request_details above), and the client would still accept the response as if it were a response to its request.

Source authentication is thus a precondition for the secure use of OSCORE and Group OSCORE. However, it is hard to provide when:

* Requests are built exclusively using shared group keying material, like in the case of a Deterministic Client.

* Requests are sent without source authentication, or their source authentication is not checked. (This was part of [I-D.ietf-core-oscore-groupcomm] in revisions before version -12)

This document does not [yet?] give full guidance on how to restore request-response binding for the general case, but currently only offers suggestions:

* The response can contain the full request. An option that allows doing that is presented in [I-D.ietf-core-responses].

* The response can contain a cryptographic hash of the full request. This is used by the method specified in this document, as defined in Section 3.3.

* The agreed-on request data can be placed in a different position in the AAD, or take part to the derivation of the (Group) OSCORE Security Context. In the latter case, care needs to be taken to never initialize a Security Context twice with the same input, as that would lead to reuse of the Authenticated Encryption with Associated Data (AEAD) nonce.

[Suggestion for any OSCORE v2: avoid request_details in the request's AAD as individual elements. Rather than having 'request_kid', 'request_piv' (and, in Group OSCORE, 'request_kid_context') as separate fields, they can better be

something more pluggable. This would avoid the need to make up an option before processing, and would allow just plugging the (hash of the) request in there, as replacing the elements for the request_details.]

Additional care has to be taken in ensuring that request_details that are not expressed in the request itself are captured. For instance, these include an indication of the Security Context through which the request is assumed to have been originated.

Requests without source authentication have to be processed assuming only the minimal possible privilege of the requester [which is currently described as the authorization of the Deterministic Client, and may be moved up here in later versions of this document]. If a response to such a request is built and it contains data more sensitive than that (which might be justified if the response is protected for an authorized group member in pairwise mode), special consideration for any side channels like response size or timing is required.

3. Deterministic Requests

This section defines a method for clients starting from a same intended CoAP request to independently build the same, corresponding Deterministic Request protected with Group OSCORE.

3.1. Deterministic Unprotected Request

When clients build their unprotected request, they can already minimize variability and thus maximize reproducibility.

This document does not set out full guidelines for minimizing the variation, but considered starting points are:

* Set the Inner CoAP Observe Option to 0 (register) [RFC7641], even if no observation is intended (and hence no Outer Observe Option is set). Thus, both Observe and non-Observe requests can be aggregated into a single request, which is upstreamed as an observation at the latest when any Observe request reaches a caching proxy.

  In this case, following a Deterministic Request that includes only an Inner Observe Option, servers include an Inner Observe Option (but no Outer Observe Option) in a successful response sent as reply. Also, when receiving a response to such a Deterministic Request previously sent, clients silently ignore the Inner Observe Option in that response.

* Avoid setting the CoAP ETag Option in requests on a whim. Instead, clients should only set it when there was a recent response conveying that ETag value. When using block-wise transfers [RFC7959] and obtaining later blocks, clients should not send the known-stale ETag value.

* When using block-wise transfers [RFC7959], maximally sized large Inner blocks (szx=6) SHOULD be selected. This serves not only to align the clients on consistent cache entries, but also helps amortize the additional data transferred in the per-message signatures.

  Outer block-wise transfer can then be used if these messages exceed a hop’s efficiently usable Maximum Transmission Unit (MTU) size.

  If Block-wise Extension for Reliable Transport (BERT) [RFC8323] is usable with OSCORE, its use is fine as well; in that case, the server picks a consistent block size for all clients anyway.

* The CoAP Padding Option defined in Appendix A can be used to limit an adversary’s ability to deduce the content and the target resource of Deterministic Requests from their length. In particular, all Deterministic Requests of the same class (ideally, all requests to a particular server) can be padded to reach the same total length, that should be agreed on among all users of the same Group OSCORE Security Context.

* Clients should not send a protected (inner) CoAP Echo Option [RFC9175] in Deterministic Requests.

  In combination with Deterministic Requests, this limits the use of the Echo Option to its inclusion as an unprotected (outer) Echo Option, and thus to testing the reachability of the client. However, this is not practically limiting, since the use as an Inner option would be to prove freshness, which is something that Deterministic Requests simply cannot provide anyway.

These guidelines only serve to ensure that cache entries are utilized. Failure to follow these guidelines has no more severe consequences than decreasing the utility and effectiveness of a cache.

3.2. Design Considerations

The hard part is determining a consensus pair (key, nonce) to be used with the AEAD cipher for encrypting the plain CoAP request and obtaining the Deterministic Request as a result, while also avoiding the reuse of the same (key, nonce) pair across different requests.

Diversity can conceptually be enforced by applying a cryptographic hash function to the complete input of the encryption operation over the plain CoAP request (i.e., the AAD and the plaintext of the COSE object), and then using the result as source of uniqueness. Any non-malleable cryptographically secure hash of sufficient length to make collisions sufficiently unlikely is suitable for this purpose.

A tempting possibility is to use a fixed (group) key, and use the hash as a deterministic AEAD nonce for each Deterministic Request through the Partial IV component (see Section 5.2 of [RFC8613]). However, the 40 bits available for the Partial IV are by far insufficient to ensure that the deterministic nonce is not reused across different Deterministic Requests. Even if the full deterministic AEAD nonce could be set, the sizes used by common algorithms would still be too small.

Consequently, the proposed method takes the opposite approach, by considering a fixed deterministic AEAD nonce, while deriving a different deterministic encryption key for each Deterministic Request. That is, the hash computed over the plain CoAP request is taken as input to the key derivation. As an advantage, this approach does not require to transport the computed hash in the CoAP OSCORE Option.

[Note: This has a further positive side effect arising with version -11 of draft-ietf-core-oscore-groupcomm. That is, since the full encoded OSCORE Option is part of the AAD, it avoids a circular dependency from feeding the AAD into the hash computation, which in turn needs crude workarounds like building the full AAD twice, or zeroing out the hash-to-be.]

3.3. The Request-Hash Option

In order to transport the hash of the plain CoAP request, a new CoAP option is defined, which MUST be supported by clients and servers that support Deterministic Requests.

The option is called Request-Hash and its properties are summarized in Table 1, which extends Table 4 of [RFC7252]. The option is Elective, Safe-to-Forward, part of the Cache-Key, and not repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD548					Request-Hash	opaque	any	(none)

Table 1: The Request-Hash Option (C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable)

The Request-Hash Option is identical in all its properties to the Request-Tag Option defined in [RFC9175], with the following exceptions:

* It is not repeatable.

* The Option Value may be arbitrarily long.

  Implementations can limit the length of the Option Value to that of the longest output of the supported hash functions.

* It may be present in responses.

  Editor’s note: Does this affect any other properties?

  A response’s Request-Hash Option is, as a matter of default value, equal to the request’s Request-Hash Option. The response is valid only if the value of its Request-Hash Option is equal to the value of the Request-Hash Option in the corresponding request.

  Servers (including proxies) thus generally should not need to include the Request-Hash Option explicitly in responses, especially as a matter of bandwidth efficiency.

  A reason (and, currently, the only known) to actually include a Request-Hash option in a response is the possible use of non-traditional responses as described in [I-D.ietf-core-responses], which in terms of that document are non-matching to the request (and thus easily usable). The Request-Hash Option included in the response allows populating caches (see below) and enables the decryption and verification of a response that is sent as a reply to a Consensus Request. In the context of non-traditional responses, the value of the Request-Hash Option in the request corresponding to a response can be inferred from the value of the Request-Hash Option in the response.

* A proxy MAY use any fresh cached response from the selected server to reply to a request with the same Request-Hash; this may save it some memory.

When replying to a request that includes a Request-Hash Option, the proxy MAY add a Request-Hash Option to the response if the option is not already present in the response, or remove the Request-Hash Option from the response if the option is already present in the response. In either case, the Request-Hash Option in the response MUST have the same value that the Request-Hash Option has in the corresponding request.

* When used with a Deterministic Request, the Request-Hash Option is created at message protection time by the sender endpoint, and it is used before message decryption and verification by the recipient endpoint. Therefore, in this use case, this option is treated as Class U for OSCORE [RFC8613] in requests. In the same application, for responses, this option is treated as Class I and is often elided from sending, in which case it is reconstructed at the recipient endpoint. Other uses of the Request-Hash Option can treat it according to different classes for the OSCORE processing.

The Request-Hash Option achieves the request-response binding described in Section 2.

3.4. Use of Deterministic Requests

This section defines how a Deterministic Request is built on the client side and then processed on the server side.

3.4.1. Preconditions

The use of Deterministic Requests in an OSCORE group requires that the interested group members are aware of the Deterministic Client in the group. In particular, they need to know:

* The Sender ID of the Deterministic Client, to be used as 'kid' parameter for the Deterministic Requests. This allows all group members to compute the Sender Key of the Deterministic Client.

  The Sender ID of the Deterministic Client is immutable throughout the lifetime of the OSCORE group. That is, it is not relinquished and it does not change upon changes of the group keying material following a group rekeying performed by the Group Manager (see Section 12.2 of [I-D.ietf-core-oscore-groupcomm]).

* The hash algorithm to use for computing the hash of a plain CoAP request, when producing the associated Deterministic Request.

Group members have to obtain this information from the Group Manager. A group member can do that, for instance, when obtaining the group keying material upon joining the OSCORE group, or later on as an active member by interacting with the Group Manager.

The joining process based on the Group Manager defined in [I-D.ietf-ace-key-groupcomm-oscore] can be easily extended to support the provisioning of information about the Deterministic Client. Such an extension is defined in Section 4 of the present document. No such extension is needed for the management interface of the Group Manager, as [I-D.ietf-ace-oscore-gm-admin] already includes the relevant parameters.

3.4.2. Client Composition of Deterministic Requests

In order to build a Deterministic Request, the client protects the plain CoAP request using the pairwise mode of Group OSCORE (see Section 8 of [I-D.ietf-core-oscore-groupcomm]), with the following alterations.

1. When preparing the OSCORE Option, the external_aad, and the AEAD nonce:

   * The Sender ID used is the Deterministic Client’s Sender ID.

   * The element ’sender_cred’ in the aad_array takes the empty CBOR byte string (0x40).

   * The Partial IV used is 0.

2. The client uses the hash function indicated for the Deterministic Client and computes a hash H over the following input: the Sender Key of the Deterministic Client, concatenated with the binary serialization of the aad_array from Step 1, concatenated with the COSE plaintext.

   Note that the payload of the plain CoAP request (if any) is not self-delimiting, and thus hash functions are limited to non-malleable ones.

3. The client derives the deterministic Pairwise Sender Key K as defined in Section 2.5.1 of [I-D.ietf-core-oscore-groupcomm], with the following differences:

   * The Sender Key of the Deterministic Client is used as the first argument of the HMAC-based Key Derivation Function (HKDF).

* The hash H from Step 2 is used as the second argument IKM-Sender of the HKDF, i.e., as a pseudo Input Keying Material (IKM) computable by all the group members.

  Note that an actual IKM-Sender cannot be obtained, since there is no authentication credential (and public key included therein) associated with the Deterministic Client to be used as Sender Authentication Credential and for computing an actual Diffie-Hellman Shared Secret.

* The Sender ID of the Deterministic Client is used as the value for the 'id' element of the 'info' parameter used as third argument of the HKDF.

4. The client includes a Request-Hash Option in the request to protect, with value set to the hash H from Step 2.

5. The client MAY include an Inner Observe Option set to 0 (register) to be protected with Group OSCORE, even if no observation is intended [RFC7641] (see Section 3.1).

6. The client protects the request using the pairwise mode of Group OSCORE as defined in Section 8.3 of [I-D.ietf-core-oscore-groupcomm], using the AEAD nonce from Step 1, the deterministic Pairwise Sender Key K from Step 3 as AEAD encryption key, and the finalized AAD.

7. The client MUST NOT include an unprotected (outer) Observe Option if no observation is intended, even in case an Inner Observe Option was included at Step 5.

8. The client MUST set 0.05 (FETCH) [RFC8132] as the Outer Code of the protected request to make it usable for a proxy's cache, even if no observation is intended.

The result is the Deterministic Request to be sent.

Since the encryption key K is derived using material from the whole plain CoAP request, this (key, nonce) pair is only used for this very message, which is deterministically encrypted unless there is a hash collision between two Deterministic Requests.

The deterministic encryption requires that the AEAD algorithm used is deterministic in itself. This is the case for all the AEAD algorithms currently registered with COSE in [COSE.Algorithms]. For future algorithms, a flag in the COSE registry is to be added.

Note that, while the process defined above is based on the pairwise mode of Group OSCORE, no information about the server takes part in the key derivation or is included in the AAD. This is intentional, since it allows sending a Deterministic Request to multiple servers at once (see Section 3.4.5). On the other hand, it requires later checks at the client when verifying a response to a Deterministic Request (see Section 3.4.4).

3.4.3. Server Processing of Deterministic Requests

Upon receiving a Deterministic Request, a server performs the following actions.

A server that does not support Deterministic Requests would not be able to create the necessary Recipient Context, and thus will fail decrypting and verifying the request.

1. If not already available, the server retrieves the information about the Deterministic Client from the Group Manager and derives the Sender Key of the Deterministic Client.

2. The server recognizes the request to be a Deterministic Request, due to the presence of the Request-Hash Option and to the 'kid' parameter of the OSCORE Option that is set to the Sender ID of the Deterministic Client.

   If the 'kid' parameter of the OSCORE Option specifies a different Sender ID than the one of the Deterministic Client, the server MUST NOT take the following steps and instead processes the request as per Section 8.4 of [I-D.ietf-core-oscore-groupcomm].

3. The server retrieves the hash H from the Request-Hash Option.

4. The server derives a Recipient Context for processing the Deterministic Request. In particular:

   * The Recipient ID is the Sender ID of the Deterministic Client.

   * The Recipient Key is derived as the key K in Step 3 of Section 3.4.2, with the hash H retrieved at Step 3 of the present Section 3.4.3.

5. The server decrypts and verifies the request using the pairwise mode of Group OSCORE as defined in Section 8.4 of [I-D.ietf-core-oscore-groupcomm] and the Recipient Context from Step 4, with the difference that the server does not perform replay checks against a Replay Window (see below).

In case of successful verification, the server MUST also perform the following actions, before possibly delivering the request to the application.

* Starting from the recovered plain CoAP request, the server MUST recompute the same hash that the client computed at Step 2 of Section 3.4.2.

  If the recomputed hash value differs from the value retrieved from the Request-Hash Option at Step 3, the server MUST treat the request as invalid and MAY reply with an unprotected 4.00 (Bad Request) error response. The server MAY set an Outer Max-Age Option with value zero. The diagnostic payload MAY contain the string "Decryption failed".

  This prevents an attacker that guessed a valid authentication tag for a given Request-Hash value to poison caches with incorrect responses.

* The server MUST verify that the unprotected request is safe to be processed in the REST sense, i.e., that it has no side effects. If verification fails, the server MUST discard the request and SHOULD reply with a protected 4.01 (Unauthorized) error response.

  Note that some CoAP implementations may not be able to prevent that an application produces side effects from a safe request. This may incur checking whether the particular resource handler is explicitly marked as eligible for processing Deterministic Requests. An implementation may also have a configured list of requests that are known to be side-effect free, or even a pre-built list of valid hashes for all sensible requests for them, and reject any other request.

  These checks replace the otherwise present requirement that the server needs to check the Replay Window of the Recipient Context (see Step 5 above), which is inapplicable with the Recipient Context derived at Step 4 from the value of the Request-Hash Option. The reasoning is analogous to the one in [I-D.amsuess-lwig-oscore] to treat the potential replay as answerable, if the handled request is side-effect free.

3.4.4. Response to a Deterministic Request

When preparing a response to a Deterministic Request, the server treats the Request-Hash Option as a Class I option. The value of the Request-Hash Option MUST be equal to the value of the Request-Hash Option that was specified in the corresponding Deterministic Request. Since the client is aware of the Request-Hash value to expect in the response, the server usually elides the Request-Hash Option from the actually transmitted response.

Treating the Request-Hash Option as a Class I option creates the request-response binding, thus ensuring that no mismatched responses can be successfully unprotected and verified by the client (see Section 2).

The client MUST reject a response to a Deterministic Request, if the value of the Request-Hash Option included in the response is not equal to the value that was specified in the Request-Hash Option of that Deterministic Request.

When preparing the response, the server performs the following actions.

1. The server includes in the response a Max-Age Option with value different from zero, thus making the Deterministic Request usable for the proxy cache.

2. The server preliminarily sets the Request-Hash Option with the full Request-Hash value, i.e., the same value of the Request-Hash Option that was specified in the Deterministic Request.

3. If the Deterministic Request included an Inner Observe Option but not an Outer Observe Option and the resource is observable [RFC7641], the server MUST include an Inner Observe Option in the response.

4. The server MUST protect the response using the group mode of Group OSCORE, as defined in Section 7.3 of [I-D.ietf-core-oscore-groupcomm]. This is required to ensure that the client can verify the source authentication of the response, since the "pairwise" key used for producing the Deterministic Request is actually shared among all the group members.

   Note that the Request-Hash Option is treated as Class I here.

5. The server MUST include its Sender Sequence Number as Partial IV in the response and use it to build the nonce to protect the response. This is required since the server does not perform replay protection on the Deterministic Request (see Section 3.4.4).

6. The server uses 2.05 (Content) as Outer Code of the response even though the response is not necessarily an Observe notification [RFC7641], in order to make the response cacheable.

7. The server SHOULD remove the Request-Hash Option from the response before sending the response to the client, as per the general option mechanism defined in Section 3.3.

8. If the Deterministic Request included an Inner Observe Option but not an Outer Observe Option, the server MUST NOT include an Outer Observe Option in the response.

Upon receiving the response, the client performs the following actions.

1. In case the response includes a 'kid' in the OSCORE Option, the client MUST verify it to be exactly the 'kid' of the server to which the Deterministic Request was sent, unless responses from multiple servers are expected (see Section 3.4.5).

2. In case the response does not include the Request-Hash Option, the client adds the Request-Hash Option to the response, setting the Option Value to the same Option Value of the Request-Hash Option that was included in the Deterministic Request.

   Otherwise, the client MUST reject the response if the Option Value of the Request-Hash Option is different from the Option Value of the Request-Hash Option that was included in the Deterministic Request.

3. The client verifies the response using the group mode of Group OSCORE, as defined in Section 7.4 of [I-D.ietf-core-oscore-groupcomm]. In particular, the client verifies the countersignature in the response, based on the 'kid' either retrieved from the OSCORE Option of the response if present therein, or otherwise of the server to which the Deterministic Request was sent to. When verifying the response, the Request-Hash Option is treated as a Class I option.

4. If the Deterministic Request included an Inner Observe Option but not an Outer Observe option (see Section 3.1), the client MUST silently ignore the Inner Observe Option in the response, which MUST NOT result in stopping the processing of the response.

   Note that this deviates from Section 4.1.3.5.2 of [RFC8613], but it is limited to a very specific situation, where the client and server both know exactly what happens. This does not affect the use of Group OSCORE in other situations.

3.4.5. Deterministic Requests to Multiple Servers

A Deterministic Request can be sent to a CoAP group, e.g., over UDP and IP multicast [I-D.ietf-core-groupcomm-bis], thus targeting multiple servers at once.

To simplify key derivation, such a Deterministic Request is still created in the same way as a one-to-one request and still protected with the pairwise mode of Group OSCORE, as defined in Section 3.4.2.

Note that this deviates from the recommendation in Section 7 of [I-D.ietf-core-oscore-groupcomm], since the Deterministic Request in this case is indeed intended to multiple recipients, but yet it is protected with the pairwise mode. However, this is limited to a very specific situation, where the client and servers both know exactly what happens. This does not affect the use of Group OSCORE in other situations.

[Note: If it was protected with the group mode, the request hash would need to be fed into a group key derivation just for this corner case. Furthermore, there would need to be a signature in spite of no authentication credential (and public key included therein) associated with the Deterministic Client.]

When a server receives a request from the Deterministic Client as addressed to a CoAP group, the server proceeds as defined in Section 3.4.3, with the difference that it MUST include its own Sender ID in the response, as the ’kid’ parameter of the OSCORE Option.

Although it is normally optional for the server to include its Sender ID when replying to a request protected in pairwise mode, it is required in this case for allowing the client to retrieve the Recipient Context associated with the server originating the response.

If a server is a member of a CoAP group, and it fails to successfully decrypt and verify an incoming Deterministic Request, then it is RECOMMENDED for that server to not reply with an error response, in case the server verifies that the Deterministic Request was sent to the CoAP group (e.g., to the associated IP multicast address) or in case the server is not able to verify that altogether.

4. Obtaining Information about the Deterministic Client

This section extends the joining process defined in [I-D.ietf-ace-key-groupcomm-oscore] and based on the Authentication and Authorization for Constrained Environments (ACE) framework [RFC9200]. Upon joining the OSCORE group, this enables a new group member to obtain from the Group Manager the required information about the Deterministic Client (see Section 3.4.1).

With reference to the 'key' parameter included in the Join Response defined in Section 6.3 of [I-D.ietf-ace-key-groupcomm-oscore], the Group_OSCORE_Input_Material object specified as its value contains also the two additional parameters 'det_senderId' and 'det_hash_alg'. These are registered in Section 7.2 of this document and are defined as follows:

* The 'det_senderId' parameter, if present, has as value the OSCORE Sender ID assigned to the Deterministic Client by the Group Manager. This parameter MUST be present if the OSCORE group uses Deterministic Requests as defined in this document. Otherwise, this parameter MUST NOT be present.

* The 'det_hash_alg' parameter, if present, has as value the hash algorithm to use for computing the hash of a plain CoAP request, when producing the associated Deterministic Request. This parameter takes values from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms]. This parameter MUST be present if the OSCORE group uses Deterministic Requests as defined in this document. Otherwise, this parameter MUST NOT be present.

The same extension above applies also to the 'key' parameter included in a Key Distribution Response (see Sections 9.1.1 and 9.1.2 of [I-D.ietf-ace-key-groupcomm-oscore]).

With reference to the 'key' parameter included in a Signature Verification Data Response defined in Section 9.6 of [I-D.ietf-ace-key-groupcomm-oscore], the Group_OSCORE_Input_Material object specified as its value contains also the 'det_senderId' parameter defined above.

5. Implementation Status

This section is to be removed before publishing as an RFC.

Note to RFC Editor: when deleting this section, please also delete RFC 7942 from the references of this document.

(Boilerplate as per Section 2.1 of [RFC7942]:)

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

5.1. Implementation in Californium

* Responsible organization: RISE Research Institutes of Sweden AB
* Implementation’s name: Cacheable OSCORE for Eclipse Californium
* Available at: https://github.com/rikard-sics/californium/tree/cacheable-oscore
* Description: Implementation in Java, building on Eclipse Californium, see:
  - https://github.com/eclipse-californium/californium
  - http://eclipse.dev/californium/
* Implementation’s level of maturity: prototype

* Version compatibility: -08 (which is the last time the wire format changed)

* Licensing: according to the same dual license of Eclipse Californium, i.e., according to the "Eclipse Distribution License 1.0" and the "Eclipse Public License 2.0". See:

  - https://github.com/eclipse-californium/californium/blob/main/LICENSE

  - https://www.eclipse.org/org/documents/edl-v10.php

  - https://www.eclipse.org/legal/epl-2.0/

* Contact point: Marco Tiloca - marco.tiloca@ri.se

* Last updated: 2025-06-27

5.2. Implementation in aiocoap

* Implementation: aiocoap https://christian.amsuess.com/tools/aiocoap/ (https://christian.amsuess.com/tools/aiocoap/)

* Description: General purpose unconstrained implementation of CoAP

* Implementation maturity: Cacheable OSCORE is part of the regular release, but not well integrated in the security setup.

* Version compatibility: -08 (which is the last time the wire format changed)

* Licensing: MIT

* Contact point: Christian Ams端ss

* Last updated: 2025-06-30

6. Security Considerations

The same security considerations from [RFC7252], [I-D.ietf-core-groupcomm-bis], [RFC8613], and [I-D.ietf-core-oscore-groupcomm] hold for this document.

The following elaborates on how, compared to Group OSCORE, Deterministic Requests dispense with some of the OSCORE security properties, by just so much as to make caching possible.

* A Deterministic Request is intrinsically designed to be replayed, as intended to be identically sent multiple times by multiple clients to the same server(s).

  Consistently, as per the processing defined in Section 3.4.3, a server receiving a Deterministic Request does not perform replay checks against an OSCORE Replay Window.

  This builds on the following considerations:

  - For a given request, the level of tolerance to replay risk is specific to the resource it operates upon (and therefore only known to the origin server). In general, if processing a request does not have state-changing side effects, the consequences of replay are not significant.

    Just like for what concerns the lack of source authentication (see below), the server must verify that the received Deterministic Request (more precisely, its handler) is side-effect free. The distinct semantics of the CoAP request methods can help the server make that assessment.

  - Consistently with the point above, a server can choose whether it will process a Deterministic Request on a per-resource basis. It is RECOMMENDED that origin servers allow resources to explicitly configure whether Deterministic Requests are appropriate to receive, as still limited to requests that are safe to be processed in the REST sense, i.e., they do not have state-changing side effects.

* Receiving a response to a Deterministic Request does not mean that the response was generated after the Deterministic Request was sent.

  However, a valid response to a Deterministic Request still contains two freshness statements:

  - It is more recent than any other response from the same group member that conveys a smaller sequence number as Partial IV.

  - It is more recent than the original creation of the deterministic keying material in the Group OSCORE Security Context.

* Source authentication of Deterministic Requests is lost.

Instead, the server must verify that the Deterministic Request (more precisely, its handler) is side-effect free. The distinct semantics of the CoAP request methods can help the server make that assessment.

Just like for what concerns the acceptance of replayed Deterministic Requests (see above), the server can choose whether it will process a Deterministic Request on a per-resource basis.

* The privacy of Deterministic Requests is limited.

  An intermediary can determine that two Deterministic Requests from different clients are identical, and thus associate the different responses generated for them.

  If a server produces responses in reply to a Deterministic Request and those responses vary in size, the server can use the Padding Option defined in Appendix A in order to hide when the response is changing.

[More on the verification of the Deterministic Request]

7. IANA Considerations

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

This document has the following actions for IANA.

7.1. CoAP Option Numbers Registry

IANA is asked to add the following entries in the "CoAP Option Numbers" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Number	Name	Reference
TBD548	Request-Hash	[RFC-XXXX]
TBD64988	Padding	[RFC-XXXX]

Table 2: Registrations in the CoAP Option Numbers Registry

[

For the Request-Hash Option, the number suggested to IANA is 548.

For the Padding Option, the option number is picked to be the highest number in the "Expert Review" range; the high option number allows it to follow practically all other options, and thus to be set when the final unpadded message length including all options is known. Therefore, the number suggested to IANA is 64988.

Applications that make use of the "Experimental use" range and want to preserve that property are invited to pick the largest suitable experimental number (65532).

Note that unless other high options are used, this means that padding a message adds an overhead of at least 3 bytes, i.e., 1 byte for the Option Delta/Length and two more bytes for the extended Option Delta (see Section 3.1 of [RFC7252]). This is considered acceptable overhead, given that the application has already chosen to prefer the privacy gains of padding over wire transfer length.

]

7.2. OSCORE Security Context Parameters Registry

IANA is asked to add the following entries in the "OSCORE Security Context Parameters" registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

* Name: det_senderId
* CBOR Label: 14 (suggested)
* CBOR Type: byte string
* Registry: -
* Description: OSCORE Sender ID assigned to the Deterministic Client of an OSCORE group
* Reference: [RFC-XXXX]

* Name: det_hash_alg
* CBOR Label: 15 (suggested)
* CBOR Type: text string / integer

* Registry: [COSE.Algorithms] Values (Hash)

* Description: Hash algorithm to use in an OSCORE group when producing a Deterministic Request

* Reference: [RFC-XXXX]

8. References

8.1. Normative References

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

[I-D.ietf-core-groupcomm-bis]
Dijk, E. and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-18, 10 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-18>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and R. H旦glund, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-28, 23 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-28>.

[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/rfc/rfc2119>.

[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/rfc/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/rfc/rfc7641>.

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

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/rfc/rfc8132>.

[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/rfc/rfc8174>.

[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/rfc/rfc8392>.

[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/rfc/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/rfc/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/rfc/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9175] Ams端ss, C., Preu 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/rfc/rfc9175>.

8.2. Informative References

[DNS-CoAP-paper]
Lenders, M. S., Amsss, C., Gndogan, C., Nawrocki, M., Schmidt, T. C., and M. Whlisch, "Securing Name Resolution in the IoT: DNS over CoAP", September 2023, <https://doi.org/10.1145/3609423>.

[I-D.amsuess-lwig-oscore]
Amsss, C., "OSCORE Implementation Guidance", Work in Progress, Internet-Draft, draft-amsuess-lwig-oscore-00, 29 April 2020, <https://datatracker.ietf.org/doc/html/draft-amsuess-lwig-oscore-00>.

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M. and F. Palombini, "Key Management for Group Object Security for Constrained RESTful Environments (Group OSCORE) Using Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-20, 25 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-20>.

[I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Hglund, R., Van der Stok, P., and F. Palombini, "Admin Interface for the OSCORE Group Manager", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-15, 23 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-oscore-gm-admin-15>.

[I-D.ietf-core-dns-over-coap]
Lenders, M. S., Amsss, C., Gndoan, C., Schmidt, T. C., and M. Whlisch, "DNS over CoAP (DoC)", Work in Progress, Internet-Draft, draft-ietf-core-dns-over-coap-20, 16 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-dns-over-coap-20>.

[I-D.ietf-core-multicast-notifications-proxy]
Tiloca, M., Hglund, R., Amsss, C., and F. Palombini, "Using Proxies for Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-multicast-notifications-proxy-00, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-multicast-notifications-proxy-00>.

[I-D.ietf-core-observe-multicast-notifications]
Tiloca, M., Hglund, R., Amsss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-13, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-observe-multicast-notifications-13>.

[I-D.ietf-core-responses]
Bormann, C. and C. Amsss, "CoAP: Non-traditional response forms", Work in Progress, Internet-Draft, draft-ietf-core-responses-00, 12 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-core-responses-00>.

[ICN-paper]
Gndoan, C., Amsss, C., Schmidt, T. C., and M. Whlisch, "Group Communication with OSCORE: RESTful Multiparty Access to a Data-Centric Web of Things", October 2021, <https://ieeexplore.ieee.org/document/9525000>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/rfc/rfc8323>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

[SW-EPIV] Lucas, G., "Star Wars", Lucasfilm Ltd. , 1977.

Appendix A. Padding

As discussed in Section 6, information can be leaked by the length of a response or, in different contexts, of a request.

In order to hide such information and mitigate the impact on privacy, the new CoAP Padding Option is defined (see Appendix A.1), as a means to increase the length of a message without changing its meaning.

The option can be used with any CoAP transport, but it is especially useful when using OSCORE or Group OSCORE, since they do not provide any padding of their own.

Before choosing to pad a message by using the Padding Option, application designers should consider whether they can arrange for common message variants to have the same length, by picking a suitable content representation; the canonical example here is expressing "yes" and "no" with "y" and "n", respectively.

A.1. The Padding Option

The option is called Padding and its properties are summarized in Table 3, which extends Table 4 of [RFC7252]. The option is Elective, Safe-to-Forward, not part of the Cache-Key, and repeatable. The option may be repeated, as that may be the only way to achieve a certain total length for the padded message.

No.	C	U	N	R	Name	Format	Length	Default
TBD64988			x	x	Padding	opaque	any	(none)

Table 3: The Padding Option (C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable)

The Padding Option is treated as Class E for OSCORE [RFC8613], which makes it indistinguishable from other Class E options or from the message payload to third parties.

A.2. Using and Processing the Padding Option

A server that produces different responses of different length for a given class of requests might wish to produce responses of consistent length, typically to hide the variation from anyone but the intended recipient. In such a case, the server can pick a length that all possible responses can be padded to, and set the Padding Option with a suitable all-zero Option Value in all responses to that class of requests.

Likewise, a client can decide on a class of requests that it pads to reach a consistent length. This has considerably less efficacy and applicability when applied to Deterministic Requests. That is: an external observer can group together different requests even if they are of the same length; and padding would hinder convergence on a single Consensus Request, thus requiring all users of the same Group OSCORE Security Context to agree in advance on the same total length for the requests.

Any party receiving a Padding Option MUST ignore it. In particular, a server MUST NOT make its choice of padding a response dependent on any padding present in the corresponding request. A means driven by the client for coordinating response padding is out of scope for this document.

Proxies that see a Padding Option MAY discard it.

Appendix B. Simple Cacheability using Ticket Requests

Building on the concept of Phantom Requests and Informative Responses defined in [I-D.ietf-core-observe-multicast-notifications], basic caching is already possible without building a Deterministic Request.

The approach discussed in this appendix is not provided for application. In fact, it is efficient only when dealing with very large representations and no OSCORE Inner block-wise mode (which is inefficient for other reasons), or when dealing with Observe notifications (which are already well covered in [I-D.ietf-core-observe-multicast-notifications]).

Rather, it is more provided as a "mental exercise" for the authors and interested readers to bridge the gap between this document and [I-D.ietf-core-observe-multicast-notifications].

That is, instead of replying to a client with a regular response, a server can send an Informative Response, defined as a protected 5.03 (Service Unavailable) error message. The payload of the Informative Response contains the Phantom Request, which is a Ticket Request in the broader terminology used by this document.

Unlike a Deterministic Request, a Phantom Request is protected with the group mode of Group OSCORE. Instead of verifying a hash, the client will be able to see from the countersignature of later responses that this was indeed the request that the server is replying to. The client also verifies that the request URI is identical between the original request and the Ticket Request.

The remaining exchange can largely play out like in [I-D.ietf-core-multicast-notifications-proxy]'s "Example with a Proxy and with Group OSCORE". That is, the client first sends the Phantom Request to the proxy (but, lacking a tp_info, without including a Listen-To-Multicast-Responses Option). Then, the proxy forwards the Phantom Request to the server, due to the lack of the Listen-To-Multicast-Responses Option.

The server then produces a regular response and includes an Outer Max-Age Option with Option Value different from zero. Note that there is no point in including an Inner Max-Age Option, as the client could not pin it in time.

When a second, different client later asks for the same resource at the same server, its new request uses a different 'kid' and 'Partial IV' than the first client's. Thus, the new request produces a cache miss at the proxy and is forwarded to the server, which replies with the same Ticket Request provided to the first client. After that, when the second client sends the Ticket Request, a cache hit at the proxy will be produced, and the Ticket Request can be served from the proxy's cache.

When multiple proxies are in use, or the response has expired from the proxy's cache, the server receives the Ticket Request multiple times. It is a matter of perspective whether the server treats that as an acceptable replay (given that this whole mechanism only makes sense on requests free of side effects), or whether it is conceptualized as having an internal proxy where the request produces a cache hit.

Appendix C. Application for More Efficient End-to-End Protected Multicast Notifications

The specification [I-D.ietf-core-observe-multicast-notifications] defines how a CoAP server can serve all clients observing a same resource at once, by sending notifications over multicast. As described in [I-D.ietf-core-multicast-notifications-proxy], the approach supports the possible presence of intermediaries such as proxies, also if Group OSCORE is used to protect notifications end-to-end.

However, comparing the "Example with a Proxy" in Section 6 of [I-D.ietf-core-multicast-notifications-proxy] and the "Example with a Proxy and with Group OSCORE" in Section 7 of [I-D.ietf-core-multicast-notifications-proxy] shows that, when using Group OSCORE, more requests need to reach the server. This is because every client originally protects its Observation request individually, and thus it needs a custom response served to obtain the Phantom Request as a Ticket Request.

If the clients send their requests as the same Deterministic Request, then the server can use these requests as Ticket Requests as well. Thus, there is no need for the server to provide a same Phantom Request to each client.

Instead, the server can send a single, unprotected Informative Response - very much like in the example without Group OSCORE - hence setting the proxy up and optionally providing also the latest notification along the way. The proxy can thus be configured by the server following the first request from the clients, after which it has an active observation and a fresh cache entry in time for the second client to arrive. This is shown by the "Example with a Proxy and with Deterministic Requests" in Section 8 of [I-D.ietf-core-multicast-notifications-proxy].

Appendix D. Open Questions

* Is "deterministic encryption" something worthwhile to consider in COSE?

  COSE would probably specify something more elaborate for the KDF (the current KDF round is the pairwise mode’s; COSE would probably run through KDF with a KDF context structure).

  COSE would give a header parameter name to the Request-Hash (which, for the purpose of OSCORE Deterministic Requests, would put back into Request-Hash by extending the option compression function across the two options).

  Conceptually, they should align well, and the implementation changes are likely limited to how the KDF is run.

* An unprotection failure from a mismatched hash will not be part of the ideally constant-time code paths that otherwise lead to AEAD unprotect failures. Is that a problem?

After all, it does tell the attacker that they did succeed in producing a valid MAC (it’s just not doing it any good, because this key is only used for Deterministic Requests and thus also needs to pass the Request-Hash check).

Appendix E. Unsorted Further Ideas

* We could allow clients to elide all other options than the Request-Hash Option, and elide the payload, if they have reason to believe that they can produce a cache hit with the abbreviated request alone.

  This may prove troublesome in terms of cache invalidation (the server would have to use short-lived responses to indicate that it does need the full request, or we would need special handling for error responses, or criteria by which proxies do not even forward these if they do not have a response at hand).

  That may be more trouble than it is worth without a strong use case (say, of complex but converging FETCH requests).

  Hashes could also be used in a truncated form for that -- should we suggest SHA-256/128 as a default? (Its birthday paradox starts to kick in at around 2^64 deterministic requests).

Appendix F. Test Vectors

This appendix includes test vectors for an example where the method defined in this document is used.

In the following, a CoAP Client C and a CoAP Server S are member of the same OSCORE group, and they exchange Deterministic Requests and corresponding responses.

Note that, while they are consistent with the presented example, the values of the Token and Message ID in the CoAP messages are only indicative, as they are subject to change throughout different message exchanges.

The considered authentication credentials of the CoAP Server S and of the Group Manager are CWT Claims Sets (CCSs) [RFC8392]. Both authentication credentials as well as the aad_array used through the Group OSCORE processing are also expressed in CBOR extended diagnostic notation as defined in Section 8 of [RFC8949] and Appendix G of [RFC8610] ("diagnostic notation").

F.1. Setup

The Group OSCORE Security Context specifies the following parameters.

* AEAD Algorithm: AES-CCM-16-64-128 (10)

* HKDF Algorithm: HKDF SHA-256 (5)

* Group Encryption Algorithm: AES-CCM-16-64-128 (10)

* Signature Algorithm: EdDSA (used with curve Ed25519) (-8, 6)

* Pairwise Key Agreement Algorithm: ECDH-SS-HKDF-256 (used with curve X25519) (-27, 4)

* Hash algorithm for Deterministic Requests: SHA-256 (-16)

* Master Secret (16 bytes):

  0x0102030405060708090a0b0c0d0e0f10

* Master Salt (8 bytes):

  0x9e7ca92223786340

* ID Context (2 bytes):

  0xdd11

* Deterministic Client’s Sender ID (1 byte):

  0xdc

* Server’s Sender ID (1 byte):

  0x52

* Server’s authentication credential as CCS (diagnostic notation):

```
{ 1: "coaps://server.example.com",
  2: "sender",
  3: "coaps://client.example.org",
  4: 1879067471,
  8: {
       1: {
            1: 1,
            3: -8,
           -1: 6,
           -2: h'77ec358c1d344e41ee0e87b8383d23a2
                 099acd39bdf989ce45b52e887463389b'
         }
     }
}
```

* Server's authentication credential as CCS (serialization) (118 bytes):

```
0xa501781a636f6170733a2f2f7365727665722e6578616d706c652e636f6d
  026673656e64657203781a636f6170733a2f2f636c69656e742e6578616d
  706c652e6f7267041a70004b4f08a101a401010327200621582077ec358c
  1d344e41ee0e87b8383d23a2099acd39bdf989ce45b52e887463389b
```

* Server's private key (serialization) (32 bytes):

```
0x857eb61d3f6d70a278a36740d132c099
  f62880ed497e27bdfd4685fa1a304f26
```

* Group Manager's authentication credential as CCS (diagnostic notation):

```
{ 1: "coaps://mysite.example.com",
  2: "groupmanager",
  3: "coaps://domain.example.org",
  4: 2879067471,
  8: {
       1: {
            1: 1,
            3: -8,
           -1: 6,
           -2: h'cde3efd3bc3f99c9c9ee210415c6cba55
                 061b5046e963b8a58c9143a61166472'
         }
     }
}
```

* Group Manager's authentication credential as CCS (serialization) (124 bytes):

```
0xa501781a636f6170733a2f2f6d79736974652e6578616d706c652e636f6d
  026c67726f75706d616e6167657203781a636f6170733a2f2f646f6d6169
  6e2e6578616d706c652e6f7267041aab9b154f08a101a401010327200621
  5820cde3efd3bc3f99c9c9ee210415c6cba55061b5046e963b8a58c9143a
  61166472
```

F.2. Deterministic Request

The client generates an unprotected CoAP GET request, which contains only the Uri-Path option with value "helloWorld". The request is Confirmable, with a Token length equal to 8 bytes.

Unprotected CoAP request (23 bytes):

```
0x48019483f0aeef1c796812a0ba68656c6c6f576f726c64
```

```
0x48 (Version: 1, Type: CON, Token Length: 8 - 1 byte)
  01 (Code: GET - 1 byte)
  9483 (Message ID - 2 bytes)
  f0aeef1c796812a0 (Token - 8 bytes)
  ba 68656c6c6f576f726c64 (Uri-path:"helloWorld" - 11 bytes)
```

The client protects the CoAP request above to produce a Deterministic Request. When doing so, the client does not include an Inner Observe option.

The following information is used to compute the Request-Hash value.

* Deterministic Client’s Sender Key (16 bytes):

  ```
  0x761c3081b8d8329790a8b321b3b4c3a4
  ```

* aad_array (diagnostic notation):

```
[1,
 [10, 10, -8, -27],
 h'dc',
 h'00',
 h'',
 h'dd11',
 h'190002dd11dc',
 h'',
 h'a501781a636f6170733a2f2f6d79736974652e6578616d706c652e636f6d
   026c67726f75706d616e6167657203781a636f6170733a2f2f646f6d6169
   6e2e6578616d706c652e6f7267041aab9b154f08a101a401010327200621
   5820cde3efd3bc3f99c9c9ee210415c6cba55061b5046e963b8a58c9143a
   61166472'
]
```

* aad_array (serialization) (150 bytes):

```
0x8901840a0a27381a41dc41004042dd1146190002dd11dc40587ca501781a
  636f6170733a2f2f6d79736974652e6578616d706c652e636f6d026c6772
  6f75706d616e6167657203781a636f6170733a2f2f646f6d61696e2e6578
  616d706c652e6f7267041aab9b154f08a101a4010103272006215820cde3
  efd3bc3f99c9c9ee210415c6cba55061b5046e963b8a58c9143a61166472
```

* Plaintext (12 bytes):

```
0x01ba68656c6c6f576f726c64
```

* Request-Hash value (32 bytes):

```
0x404b3a7c9f8c878a0b5246cca71e3926
  f0a8cebefdcabbc80e79579d5a1ee17d
```

The following information is used to produce the protected Deterministic Request.

* Partial IV (1 byte):

```
0x00
```

* kid (1 byte):

```
0xdc
```

* kid context (2 bytes):

```
0xdd11
```

* AAD (serialization) (163 bytes):

  0x8368456e6372797074304058968901840a0a27381a41dc41004042dd114619
  0002dd11dc40587ca501781a636f6170733a2f2f6d79736974652e6578616d
  706c652e636f6d026c67726f75706d616e6167657203781a636f6170733a2f
  2f646f6d61696e2e6578616d706c652e6f7267041aab9b154f08a101a40101
  03272006215820cde3efd3bc3f99c9c9ee210415c6cba55061b5046e963b8a
  58c9143a61166472

* Plaintext (12 bytes):

  0x01ba68656c6c6f576f726c64

* Encryption Key (16 bytes):

  0x5d4671f0d12d27a59dec68c2e2ebcc88

* Nonce (13 bytes):

  0x46eb80969ab7389b084dd6f996

From the previous parameter, the following is derived:

* OSCORE option value (6 bytes):

  0x190002dd11dc

* Request-Hash option value (32 bytes):

  0x404b3a7c9f8c878a0b5246cca71e3926
  f0a8cebefdcabbc80e79579d5a1ee17d

* Ciphertext (20 bytes):

  0x65bcd5d5edf413497bfdeec8975e2acafa702b45

From there, the protected CoAP request as Deterministic Request (76 bytes):

```
0x48059483f0aeef1c796812a096190002dd11dced010e13404b3a7c9f8c878a
  0b5246cca71e3926f0a8cebefdcabbc80e79579d5a1ee17dff65bcd5d5edf4
  13497bfdeec8975e2acafa702b45

0x48 (version:1, type: CON, Token Length: 8 - 1 byte)
  05 (Code: FETCH - 1 byte)
  9483 (Message ID - 2 bytes)
  f0aeef1c796812a0 (Token - 8 bytes)
  96 190002dd11dc (OSCORE option - 7 bytes)
  ed 010e 13 404b3a7c9f8c878a0b5246cca71e3926f0a8cebefdcabbc8
             0e79579d5a1ee17d (Request-Hash option - 36 bytes)
  ff (payload marker - 1 byte)
  65bcd5d5edf413497bfdeec8975e2acafa702b45 (payload - 20 bytes)
```

F.3. Response to Deterministic Request

The server responds to the first request with an ACK message, to which a 2.05 (Content) response indicating a Max-Age of 3600 seconds is piggybacked. The payload of the response is the ASCII-encoded string ". ID: 42".

Unprotected CoAP response (61 bytes):

```
0x68459483f0aeef1c796812a0d2010e10ed010913404b3a7c9f8c878a0b5246
  cca71e3926f0a8cebefdcabbc80e79579d5a1ee17dff2e2049443a203432

0x68 (version: 1, type: ACK, Token Length: 8 - 1 byte)
  45 (Code: 2.05 - 1 byte)
  9483 (Message ID - 2 bytes)
  f0aeef1c796812a0 (Token - 8 bytes)
  d2 01 0e10 (Max-Age:3600 - 4 bytes)
  ed 0109 13 404b3a7c9f8c878a0b5246cca71e3926f0a8cebefdcabbc8
             0e79579d5a1ee17d (Request-Hash - 36 bytes)
  ff (payload marker - 1 byte)
  2e2049443a203432 (payload - 8 bytes)
```

The server protects the CoAP response above as follows. When doing so, the server: does not include an Inner Observe option; includes its own Sender ID in the ’kid’ of the OSCORE option; elides the Request-Hash option from the response.

The following information is used to protect the response.

* Partial IV (1 byte):

  0x00

* kid (1 byte):

  0x52

* kid context (2 bytes):

  0xdd11

* aad_array (diagnostic notation):

```
[1,
 [10, 10, -8, -27],
 h'dc',
 h'00',
 h'ed011713404b3a7c9f8c878a0b5246cca71e3926f0a8cebefdcabbc80e79
   579d5a1ee17d',
 h'dd11',
 h'290052',
 h'a501781a636f6170733a2f2f7365727665722e6578616d706c652e636f6d
   026673656e64657203781a636f6170733a2f2f636c69656e742e6578616d
   706c652e6f7267041a70004b4f08a101a401010327200621582077ec358c
   1d344e41ee0e87b8383d23a2099acd39bdf989ce45b52e887463389b',
 h'a501781a636f6170733a2f2f6d79736974652e6578616d706c652e636f6d
   026c67726f75706d616e6167657203781a636f6170733a2f2f646f6d6169
   6e2e6578616d706c652e6f7267041aab9b154f08a101a401010327200621
   5820cde3efd3bc3f99c9c9ee210415c6cba55061b5046e963b8a58c9143a
   61166472'
]
```

* aad_array (serialization) (303 bytes):

```
0x8901840a0a27381a41dc41005824ed011713404b3a7c9f8c878a0b5246cc
  a71e3926f0a8cebefdcabbc80e79579d5a1ee17d42dd11432900525876a5
  01781a636f6170733a2f2f7365727665722e6578616d706c652e636f6d02
  6673656e64657203781a636f6170733a2f2f636c69656e742e6578616d70
  6c652e6f7267041a70004b4f08a101a401010327200621582077ec358c1d
  344e41ee0e87b8383d23a2099acd39bdf989ce45b52e887463389b587ca5
  01781a636f6170733a2f2f6d79736974652e6578616d706c652e636f6d02
  6c67726f75706d616e6167657203781a636f6170733a2f2f646f6d61696e
  2e6578616d706c652e6f7267041aab9b154f08a101a40101032720062158
  20cde3efd3bc3f99c9c9ee210415c6cba55061b5046e963b8a58c9143a61
  166472
```

* AAD (serialization) (317 bytes):

```
0x8368456e6372797074304059012f8901840a0a27381a41dc41005824ed01
  1713404b3a7c9f8c878a0b5246cca71e3926f0a8cebefdcabbc80e79579d
  5a1ee17d42dd11432900525876a501781a636f6170733a2f2f7365727665
  722e6578616d706c652e636f6d026673656e64657203781a636f6170733a
  2f2f636c69656e742e6578616d706c652e6f7267041a70004b4f08a101a4
  0101032720062158207ec358c1d344e41ee0e87b8383d23a2099acd39bd
  f989ce45b52e887463389b587ca501781a636f6170733a2f2f6d79736974
  652e6578616d706c652e636f6d026c67726f75706d616e6167657203781a
  636f6170733a2f2f646f6d61696e2e6578616d706c652e6f7267041aab9b
  154f08a101a401010327200621582cde3efd3bc3f99c9c9ee210415c6cb
  a55061b5046e963b8a58c9143a61166472
```

* Plaintext (14 bytes):

```
0x45d2010e10ff2e2049443a203432
```

* Encryption Key (16 bytes):

```
0xa8c8b7db5d05cfc7faa2bb1afaca6c2f
```

* Nonce (13 bytes):

```
0x46eb80969ab73815084dd6f996
```

From the previous parameter, the following is derived:

* OSCORE option value (3 bytes):

```
0x290052
```

* Request-Hash option value (32 bytes):

```
0x404b3a7c9f8c878a0b5246cca71e3926
  f0a8cebefdcabbc80e79579d5a1ee17d
```

* Ciphertext (22 bytes):

```
0xcbc7356c84c10b626fef8bd57ed2dfaeec175f8e44e1
```

* Plain signature (64 bytes):

```
0x44879f32ab6e8533fd89dedada6e104d10d88ea42fa6d0
  c8e7ccb21e0088e0226ef98405a84f13525a22fd7cf327
  9f3b1cee59af3f45e96d38c48f38a0217405
```

* Signature Encryption Key (16 bytes):

```
      0x85ca7c0bc5b8ea2e267b203dc3b71ce6
```

* Signature Keystream (64 bytes):

```
      0xf6161f5ea4d6375819924cbcac00f45a469c517f0a435d
        e590b7a242064d5afc092fa2290b480166701088a1c4ad
        06d3e933a3f21a39b3204f7159b977193ce8
```

* Encrypted Signature (64 bytes):

```
      0xb291806c0fb8b26be41b9266766ee4175644dfdb25e58d
        2d777b105c06c5bade67d6262ca30712342a3275dd378a
        99e8f5ddfa5d257c5a4d77b5d681d73848ed
```

From there, the protected CoAP response (106 bytes):

```
0x68459483f0aeef1c796812a093290052520e10ffcbc7356c84c10b626fef8b
  d57ed2dfaeec175f8e44e1b291806c0fb8b26be41b9266766ee4175644dfdb
  25e58d2d777b105c06c5bade67d6262ca30712342a3275dd378a99e8f5ddfa
  5d257c5a4d77b5d681d73848ed

0x68 (version:1, type: ACK, Token Length: 8 - 1 byte)
  45 (Code: 2.05 - 1 byte)
  9483 (Message ID - 2 bytes)
  f0aeef1c796812a0 (Token - 8 bytes)
  93 290052 (OSCORE option - 4 bytes)
  52 0e10 (Max age option - 3 bytes)
  ff (payload marker - 1 byte)
  cbc7356c84c10b626fef8bd57ed2dfaeec175f8e44e1b29180
    6c0fb8b26be41b9266766ee4175644dfdb25e58d2d777b10
    5c06c5bade67d6262ca30712342a3275dd378a99e8f5ddfa
    5d257c5a4d77b5d681d73848ed (payload - 86 bytes)
```

Appendix G. Document Updates

This section is to be removed before publishing as an RFC.

G.1. Version -00 to -01

* Updated title.

* Revised abstract and first part of the introduction.

* Updated references.

* Suggested values for IANA registrations.

* Removed changelog for the individual submission document.

* Minor clarifications and editorial improvements.

Acknowledgments

The authors sincerely thank Martine Lenders, Michael Richardson, Jim Schaad, and Gran Selander for their comments and feedback.

This work was supported by the Sweden’s Innovation Agency VINNOVA within the EUREKA CELTIC-NEXT projects CRITISEC and CYPRESS; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors’ Addresses

Christian Amsss
Austria
Email: christian@amsuess.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se