Workgroup: CoRE Working Group
Internet-Draft:
draft-amsuess-core-cachable-oscore-06
Published: 11 January 2023
Intended Status: Standards Track
Expires: 15 July 2023
Authors: C. Amsüss M. Tiloca
RISE AB

Cacheable OSCORE

Abstract

Group communication with the Constrained Application Protocol (CoAP) can be secured end-to-end using Group Object Security for Constrained RESTful Environments (Group OSCORE), also across untrusted intermediary proxies. However, this sidesteps the proxies' abilities to cache responses from the origin server(s). This specification restores cacheability of protected responses at proxies, by introducing consensus requests which any client in a 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 CORE 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://gitlab.com/chrysn/core-cachable-oscore/.

Status of This Memo

Table of Contents

1. Introduction
  1.1. Use cases
  1.2. Terminology
2. OSCORE processing without source authentication
3. Deterministic Requests
  3.1. Deterministic Unprotected Request
  3.2. Design Considerations
  3.3. Request-Hash
  3.4. Use of Deterministic Requests
    3.4.1. Pre-Conditions
    3.4.2. Client Processing of Deterministic Request
    3.4.3. Server Processing of Deterministic Request
    3.4.4. Response to a Deterministic Request
    3.4.5. Deterministic Requests to Multiple Servers
4. Obtaining Information about the Deterministic Client
5. Security Considerations
6. IANA Considerations
  6.1. CoAP Option Numbers Registry
  6.2. OSCORE Security Context Parameters Registry
7. References
  7.1. Normative References
  7.2. Informative References
Appendix A. Change log
Appendix B. Padding
  B.1. Definition of the Padding Option
  B.2. Using and processing the Padding option
Appendix C. Simple Cacheability using Ticket Requests
Appendix D. Application for More Efficient End-to-End Protected Multicast Notifications
Appendix E. Open questions
Appendix F. Unsorted further ideas
Acknowledgments

Authors' Addresses

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports also group communication, for instance over UDP and IP multicast [I-D.ietf-core-groupcomm-bis]. In a group communication environment, exchanged messages can be secured end-to-end by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm].

Requests and responses protected with the group mode of Group OSCORE can be read by all group members, i.e., not only by the intended recipient(s), thus achieving group-level confidentiality.

This allows a trusted intermediary proxy which is also a member of the OSCORE group to populate its cache with responses from origin servers. Later on, the proxy can possibly reply to a request in the group with a response from its cache, if recognized as an eligible server by the client.

However, an untrusted proxy which is not member of the OSCORE group only sees protected responses as opaque, uncacheable ciphertext. In particular, different clients in the group that originate a same plain CoAP request would send different protected requests, as a result of their Group OSCORE processing. Such protected requests cannot yield a cache hit at the proxy, which makes the whole caching of protected responses pointless.

This document addresses this complication and enables cacheability of protected responses, also for proxies that are not members of the OSCORE group and are unaware of 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 defines "Deterministic Request" as a convenient incarnation of such concept.

All clients wishing to send a particular GET or FETCH request are able to deterministically compute the same protected request, using a variation on the pairwise mode of Group OSCORE. 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 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 in two halves:

* maintaining request-response bindings in 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-observe-multicast-notifications], 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 D).

When relying on Information-Centric Networking (ICN) for multiparty dissemination of cacheable content, CoAP and CoAP proxies can be used to enable asynchronous group communication. This leverages CoAP proxies performing request aggregation, as well as response replication and cacheability [ICN-paper]. By restoring cacheability of OSCORE-protected responses, the Deterministic Requests defined in this document make it possible to attain dissemination of cacheable content in ICN-based deployments, also when the content is protected end-to-end.

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], COSE [RFC9052][RFC9053], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This document 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 response to it (for example to tell which path was requested), 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 sets up the Deterministic Client, and assigns it a unique Sender ID as 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 a Deterministic Request. The prototypical Ticket Request is the Phantom Request defined in [I-D.ietf-core-observe-multicast-notifications].

  In Appendix C, the term is used to bridge the gap to that draft.

2. OSCORE processing without source authentication

The request-response binding of OSCORE is achieved by the request_kid / request_piv items (and, in group OSCORE, request_kid_context) present in the response's AAD. Its security depends on the server obtaining source authentication for the request: Without, 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 to secure use of OSCORE. However, it is hard to provide when:

* Requests are built exclusively using shared key material (as in a Deterministic Client).

* Requests are sent without source authentication, or where the source authentication is not checked. (This was part of [I-D.ietf-core-oscore-groupcomm] in revisions before -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 was presented in [I-D.bormann-core-responses].

* The response can contain a cryptographic hash of the full request. This is used in Section 3.3.

* The above details can be transported in a Class E option (encrypted) or a a Class I option (unencrypted but part of the AAD). The latter has the advantage that it can be removed in transit and reconstructed at the receiver.

* Alternatively, the agreed-on request data can be placed in a different position in the AAD, or be part of the security context derivation. 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 nonce reuse.

[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 in the hash or request in there replacing the request_ items.]

Additional care has to be taken that details not expressed in the request itself (like the security context from which it is assumed to have originated) are captured.

Processing of requests without source authentication has to be done assuming only the minimal possible privilege of the requester [which currently described as the authorization of the Deterministic Client, and may be moved up here in later versions of this document]. If a response is built to such a request that contains data more sensitive than that (which might be justified if the response is protected for an authorized group member in pairwise response 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 plain CoAP request to independently arrive at a same Deterministic Request protected with Group OSCORE.

3.1. Deterministic Unprotected Request

Clients build the unprotected Deterministic Request in a way which is as much reproducible as possible. This document does not set out full guidelines for minimizing the variation, but considered starting points are:

* Set the inner Observe option to 0 even if no observation is intended (and hence no outer Observe is set). Thus, both observing and non-observing requests can be aggregated into a single request, that is upstreamed as an observation at the latest when any observing 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 have to silently ignore the inner Observe option in that response.

* Avoid setting the ETag option in requests on a whim. Only set it when there was a recent response with that ETag. When obtaining later blocks, do not send the known-stale ETag.

* In block-wise transfer, 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 excede a hop's efficiently usable MTU size.

(If 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 Padding option defined in Appendix B 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 OSCORE Security Context.

* Clients should not send any inner Echo options [RFC9175] in Deterministic Requests.

  This limits the use of the Echo option in combination with Deterministic Requests to unprotected (outer) options, and thus is limited to testing the reachability of the client. This is not practically limiting, as the use as an inner option would be to prove freshness, which is something Deterministic Requests simply cannot provide anyway.

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

3.2. Design Considerations

The hard part is arriving at a consensus pair (key, nonce) to be used with the AEAD cipher for encrypting the Deterministic Request, while also avoiding 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 throught the Partial IV component (see Section 5.2 of [RFC8613]). However, the 40 bit 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.

As a consequence, the proposed method takes the opposite approach, by considering a fixed deterministic AEAD nonce, while generating 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 generation. As an advantage, this approach does not require to transport the computed hash in the OSCORE option.

[Note: This has a further positive side effect arising with version -11 of Group OSCORE. 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. Request-Hash

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. As summarized in Figure 1, the Request-Hash option is elective, safe to forward, part of the cache key and repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD1				x	Request-Hash	opaque	any	(none)

Figure 1: Request-Hash Option

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

- It may be arbitrarily long.

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

- It may be present in responses (TBD: Does this affect any other properties?).

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

Servers (including proxies) thus generally SHOULD NOT need to send the Request-Hash option explicitly in responses, especially as a matter of bandwidth efficiency.

A reason (and, currently, the only known) to actually send a Request-Hash in a response are non-traditional responses as described in [I-D.bormann-core-responses], which in terms of that document are non-matching to the request (and thus easily usable); the request hash in the response allows populating caches (see below) and decryption of the response in Deterministic Request contexts. In the context of non-traditional responses, a matching request's Request-Hash can be inferred from its value in the response.

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

  A proxy can add or remove the request's Request-Tag value to / from a response.

* When used with a Deterministic Request, this option is created at message protection time by the sender, and used before message unprotection by the recipient. Therefore, in this use case, it is treated as Class U for OSCORE [RFC8613] in requests. In the same application, for responses, it is treated as Class I, and often elided from sending (but reconstructed at the receiver). Other uses of this option can put it into different classes for the OSCORE processing.

This option achieves 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. Pre-Conditions

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.

- 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 sending a request to a dedicated resource at 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 this document.

3.4.2. Client Processing of Deterministic Request

In order to build a Deterministic Request, the client protects the plain CoAP request using the pairwise mode of Group OSCORE (see Section 9 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 used Sender ID is the Deterministic Client's Sender ID.

   - The used Partial IV is 0.

   When preparing the external_aad, the element 'sender_public_key' in the aad_array takes the empty CBOR byte string.

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 external_aad 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 2.3.1 of [I-D.ietf-core-oscore-groupcomm], with the following differences:

   * The Sender Key of the Deterministic Client is used as first argument of the HKDF.

   * The hash H from step 2 is used as second argument of the HKDF, i.e., as a pseudo IKM-Sender 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 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 to be protected with OSCORE, even if no observation is intended (see Section 3.1).

6. The client protects the request using the pairwise mode of Group OSCORE as defined in Section 9.3 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 above.

8. The client sets FETCH as the outer code of the protected request to make it usable for a proxy's cache, even if no observation is requested [RFC7641].

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 the used AEAD algorithm to be 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 to the key derivation or is included in the AAD. This is intentional, since it allows for 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 Request

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 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 actually 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 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 9.4 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 the previous step.

5. The server verifies the request using the pairwise mode of Group OSCORE, as defined in Section 9.4 of [I-D.ietf-core-oscore-groupcomm], using 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 message 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 treating a response to a Deterministic Request, the Request-Hash option is treated as a Class I option (but usually not sent). This creates the request-response binding ensuring that no mismatched responses can be successfully unprotected (see Section 2). The client MUST reject responses with a Request-Hash not matching the one it sent in the request.

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

1. The server sets a non-zero Max-Age option, thus making the Deterministic Request usable for the proxy cache.

2. The server preliminarily sets the Request-Hash option with the full request hash.

3. If the Deterministic Request included an inner Observe option but not an outer Observe option, 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 8.3 of [I-D.ietf-core-oscore-groupcomm]. This is required to ensure that the client can verify source authentication of the response, since the "pairwise" key used for 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 use its own Sender Sequence Number as Partial IV to protect the response, and include it as Partial IV in the OSCORE option of 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 even though it 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 message before sending as per the general option mechanism of 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 and unless responses from multiple servers are expected (see Section 3.4.5), the client MUST verify it to be exactly the 'kid' of the server to which the Deterministic Request was sent.

2. The client sets the Request-Hash option with the full request hash on the reponse. If an option of a different value is already present, it rejects the response.

3. The client verifies the response using the group mode of Group OSCORE, as defined in Section 8.4 of [I-D.ietf-core-oscore-groupcomm]. In particular, the client verifies the counter signature in the response, based on the 'kid' of the server it sent the request 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 and MUST NOT stop processing the response.

[Note: This deviates from Section 4.1.3.5.2 of RFC 8613, 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 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 Section 8 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 '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 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 send back any error message, in case the server asserts 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 assert 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 ACE framework for Authentication and Authorization [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 of the Joining Response defined in Section 6.4 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 defined in Section 6.2 of this document. In particular:

- 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 when included in a Key Distribution Response (see Sections 8.1 and 8.2 of [I-D.ietf-ace-key-groupcomm-oscore]) and in a Signature Verification Data Response (see Section 13 of [I-D.ietf-ace-key-groupcomm-oscore]).

5. Security Considerations

The same security considerations from [RFC7252] [I-D.ietf-core-groupcomm-bis][RFC8613] [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's 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 (precisely: its handler) is side effect free. The distinct semantics of the CoAP request codes 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 has a smaller sequence number.

  - It is more recent than the original creation of the deterministic security context's key material.

* Source authentication of Deterministic Requests is lost.

  Instead, the server must verify that the Deterministic Request (precisely: its handler) is side effect free. The distinct semantics of the CoAP request codes 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 associate the different responses generated for them. A server producing responses of varying size to a Deterministic Request can use the Padding option to hide when the response is changing.

[More on the verification of the Deterministic Request]

6. IANA Considerations

This document has the following actions for IANA.

6.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry defined in [RFC7252] within the "CoRE Parameters" registry.

Number	Name	Reference
TBD1	Request-Hash	[[this document]]
TBD2	Padding	[[this document]]

Figure 2: CoAP Option Numbers

[

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 Experts 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 option delta/length and two more bytes of extended option delta. This is considered acceptable overhead, given that the application has already chosen to prefer the privacy gains of padding over wire transfer length.

]

6.2. OSCORE Security Context Parameters Registry

IANA is asked to register the following entries in the "OSCORE Security Context Parameters" Registry defined in Section 9.4 of [RFC9203].

*Name: det_senderId

*CBOR Label: TBD3

*CBOR Type: bstr

*Registry: -

*Description: OSCORE Sender ID assigned to the Deterministic Client of an OSCORE group

*Reference: [[this document]] (Section 4)

*Name: det_hash_alg

*CBOR Label: TBD4

*CBOR Type: int / tstr

*Registry: -

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

*Reference: [[this document]] (Section 4)

7. References

7.1. Normative References

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

[I-D.ietf-core-groupcomm-bis] Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-07, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-07.txt>.

[I-D.ietf-core-oscore-groupcomm] Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-17, 20 December 2022, <https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-17.txt>.

[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>.

[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>.

[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/info/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/info/rfc8174>.

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

[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/info/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/info/rfc9053>.

7.2. Informative References

[I-D.amsuess-lwig-oscore] Amsüss, C., "OSCORE Implementation Guidance", Work in Progress, Internet-Draft, draft-amsuess-lwig-oscore-00, 29 April 2020, <https://www.ietf.org/archive/id/draft-amsuess-lwig-oscore-00.txt>.

[I-D.bormann-core-responses] Bormann, C. and C. Amsüss, "CoAP: Non-traditional response forms", Work in Progress, Internet-Draft, draft-bormann-core-responses-01, 3 February 2022, <https://www.ietf.org/archive/id/draft-bormann-core-responses-01.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-15, 5 September 2022, <https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-15.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-05, 24 October 2022, <https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-05.txt>.

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

[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>.

[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/info/rfc8323>.

[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>.

[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/info/rfc9200>.

[RFC9203] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "The Object Security for Constrained RESTful Environments (OSCORE) Profile of the Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9203, DOI 10.17487/RFC9203, August 2022, <https://www.rfc-editor.org/info/rfc9203>.

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

Appendix A. Change log

Since -05:

* Updated references.

Since -04:

* Revised and extended list of use cases.

* Added further note on Deterministic Requests to a group of servers as still protected with the pairwise mode.

* Suppression of error responses for servers in a CoAP group.

* Extended security considerations with discussion on replayed requests.

Since -03:

* Processing steps in case only inner Observe is included.

* Clarified preserving/eliding the Request-Hash option in responses.

* Clarified limited use of the Echo option.

* Clarifications on using the Padding option.

Since -02:

* Separate parts needed to respond to unauthenticated requests from the remaining deterministic response part. (Currently this is mainly an addition; the document will undergo further refactoring if that split proves helpful).

* Inner Observe is set unconditionally in Deterministic Requests.

* Clarifications around padding and security considerations.

Since -01:

* Not meddling with request_kid any more.

  Instead, Request-Hash in responses is treated as Class I, but typically elided.

  In requests, this removes the need to compute the external_aad twice.

* Derivation of the hash now uses the external_aad, rather than the full AAD. This is good enough because AAD is a function only of the external_aad, and the external_aad is easier to get your hands on if COSE manages all the rest.

* The Sender ID of the Deterministic Client is immutable throughout the group lifetime. Hence, no need for any related expiration/creation time and mechanisms to perform its update in the group.

* Extension to the ACE Group Manager of ace-key-groupcomm-oscore to provide required info about the Deterministic Client to new group members when joining the group.

* Alignment with changes in core-oscore-groupcomm-12.

* Editorial improvements.

Since -00:

* More precise specification of the hashing (guided by first implementations)

* Focus shifted to Deterministic Requests (where it should have been in the first place; all the build-up of Token Requests was moved to a motivating appendix)

* Aligned with draft-tiloca-core-observe-responses-multicast-05 (not submitted at the time of submission)

* List the security properties lost compared to OSCORE

Appendix B. Padding

As discussed in Section 5, 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 following Padding option is defined, to allow increasing a message's length without changing its meaning.

The option can be used with any CoAP transport, but is especially useful with OSCORE as that does not provide any padding of its 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.

B.1. Definition of the Padding Option

As summarized in Figure 3, the Padding option is elective, safe to forward and not part of the cache key; these follow from the usage instructions. 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
TBD2			x	x	Padding	opaque	any	(none)

Figure 3: Padding Option

When used with OSCORE, the Padding option is of Class E, this makes it indistinguishable from other Class E options or the payload to third parties.

B.2. Using and processing the Padding option

When a server produces different responses of different length for a given class of requests but wishes to produce responses of consistent length (typically to hide the variation from anyone but the intended recipient), 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 consistent length. This has considerably less efficacy and applicability when applied to Deterministic Requests. That is: an external observer can group 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 OSCORE Security Context to agree on the same total length in advance.

Any party receiving a Padding option MUST ignore it. In particular, a server MUST NOT make its choice of padding dependent on any padding present in the request. (An option to coordinate response padding driven by the client is out of scope for this document).

Proxies that see a padding option MAY discard it.

Appendix C. 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 this document's broader terminology.

Unlike a Deterministic Request, a Phantom Request is protected in Group Mode. Instead of verifying a hash, the client can see from the signature that this was indeed the request the server is answering. The client also verifies that the request URI is identical between the original request and the Ticket Request.

The remaining exchange largely plays out like in [I-D.ietf-core-observe-multicast-notifications]'s "Example with a Proxy and Group OSCORE": The client sends the Phantom Request to the proxy (but, lacking a tp_info, without a Listen-To-Multicast-Responses option), which forwards it to the server for lack of the option.

The server then produces a regular response and includes a non-zero Max-Age option as an outer CoAP option. 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 responds 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 mechansim 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 D. Application for More Efficient End-to-End Protected Multicast Notifications

[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. 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 Appendix E of [I-D.ietf-core-observe-multicast-notifications] and the "Example with a Proxy and Group OSCORE" in Appendix F of [I-D.ietf-core-observe-multicast-notifications] shows that, when using Group OSCORE, more requests need to hit the server. This is because every client originally protects its Observation request individually, and thus 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, 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.

Appendix E. 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 F. Unsorted further ideas

* All or none of the Deterministic Requests should have an inner observe option. Preferably none -- that makes messages shorter, and clients need to ignore that option either way when checking whether a Consensus Request matches their intended request.

* We could allows clients to elide all other options than Request-Hash, and elide the payload, if it has reason to believe that it 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'd need special handling for error responses, or criteria by which proxies don't even forward these if they don't have a response at hand).

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

  Hashes could also be used in truncated form for that.

Acknowledgments

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

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Christian Amsüss
Austria

Email: christian@amsuess.com

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

Email: marco.tiloca@ri.se