CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: May 7, 2020

C. Amsuess
J. Mattsson
G. Selander
Ericsson AB

November 04, 2019

CoAP: Echo, Request-Tag, and Token Processing
draft-ietf-core-echo-request-tag-08

Abstract

This document specifies enhancements to the Constrained Application Protocol (CoAP) that mitigate security issues in particular use cases. The Echo option enables a CoAP server to verify the freshness of a request or to force a client to demonstrate reachability at its claimed network address. The Request-Tag option allows the CoAP server to match block-wise message fragments belonging to the same request. The update to the client Token processing requirements of RFC 7252 forbids non-secure reuse of Tokens to ensure binding of responses to requests when CoAP is used with security.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 7, 2020.

Table of Contents

1. Introduction

The initial Constrained Application Protocol (CoAP) suite of specifications ([RFC7252], [RFC7641], and [RFC7959]) was designed with the assumption that security could be provided on a separate layer, in particular by using DTLS ([RFC6347]). However, for some use cases, additional functionality or extra processing is needed to support secure CoAP operations. This document specifies security enhancements to the Constrained Application Protocol (CoAP).

This document specifies two CoAP options, the Echo option and the Request-Tag option: The Echo option enables a CoAP server to verify the freshness of a request, synchronize state, or force a client to demonstrate reachability at its claimed network address. The Request-Tag option allows the CoAP server to match message fragments belonging to the same request, fragmented using the CoAP block-wise Transfer mechanism, which mitigates attacks and enables concurrent block-wise operations. These options in themselves do not replace the need for a security protocol; they specify the format and processing of data which, when integrity protected using e.g. DTLS ([RFC6347]), TLS ([RFC8446]), or OSCORE ([RFC8613]), provide the additional security features.

The document also updates the Token processing requirements for clients specified in [RFC7252]. The updated processing forbids non-secure reuse of Tokens to ensure binding of responses to requests when CoAP is used with security, thus mitigating error cases and attacks where the client may erroneously associate the wrong response to a request.

Each of the following sections provides a more detailed introduction to the topic at hand in its first subsection.

1.1. Terminology

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

Unless otherwise specified, the terms "client" and "server" refers to "CoAP client" and "CoAP server", respectively, as defined in [RFC7252]. The term "origin server" is used as in [RFC7252]. The term "origin client" is used in this document to denote the client from which a request originates; to distinguish from clients in proxies.

The terms "payload" and "body" of a message are used as in [RFC7959]. The complete interchange of a request and a response body is called a (REST) "operation". An operation fragmented using [RFC7959] is called a "block-wise operation". A block-wise operation which is fragmenting the request body is called a "block-wise request operation". A block-wise operation which is fragmenting the response body is called a "block-wise response operation".

Two request messages are said to be "matchable" if they occur between the same endpoint pair, have the same code and the same set of options except for elective NoCacheKey options and options involved in block-wise transfer (Block1, Block2 and Request-Tag). Two operations are said to be matchable if any of their messages are.

Two matchable block-wise operations are said to be "concurrent" if a block of the second request is exchanged even though the client still intends to exchange further blocks in the first operation. (Concurrent block-wise request operations from a single endpoint are impossible with the options of [RFC7959] (see the last paragraphs of Sections 2.4 and 2.5) because the second operation's block overwrites any state of the first exchange.).

The Echo and Request-Tag options are defined in this document.

2. Request Freshness and the Echo Option

2.1. Request Freshness

A CoAP server receiving a request is in general not able to verify when the request was sent by the CoAP client. This remains true even if the request was protected with a security protocol, such as DTLS. This makes CoAP requests vulnerable to certain delay attacks which are particularly perilous in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks can be mitigated by establishing fresh session keys, e.g. performing a DTLS handshake for each request, but in general this is not a solution suitable for constrained environments, for example, due to increased message overhead and latency. Additionally, if there are proxies, fresh DTLS session keys between server and proxy does not say anything about when the client made the request. In a general hop-by-hop setting, freshness may need to be verified in each hop.

A straightforward mitigation of potential delayed requests is that the CoAP server rejects a request the first time it appears and asks the CoAP client to prove that it intended to make the request at this point in time.

2.2. The Echo Option

This document defines the Echo option, a a lightweight challenge-response mechanism for CoAP that enables a CoAP server to verify the freshness of a request. A fresh request is one whose age has not yet exceeded the freshness requirements set by the server. The freshness requirements are application specific and may vary based on resource, method, and parameters outside of CoAP such as policies. The Echo option value is a challenge from the server to the client included in a CoAP response and echoed back to the server in one or more CoAP requests. The Echo option provides a convention to transfer freshness indicators that works for all CoAP methods and response codes.

This mechanism is not only important in the case of actuators, or other use cases where the CoAP operations require freshness of requests, but also in general for synchronizing state between CoAP client and server, cryptographically verify the aliveness of the client, or force a client to demonstrate reachability at its claimed network address. The same functionality can be provided by echoing freshness indicators in CoAP payloads, but this only works for methods and response codes defined to have a payload. The Echo option provides a convention to transfer freshness indicators that works for all methods and response codes.

2.2.1. Echo Option Format

The Echo Option is elective, safe-to-forward, not part of the cache-key, and not repeatable, see Figure 1, which extends Table 4 of [RFC7252]).

No.	C	U	N	R	Name	Format	Len.	Default	E	U
TBD			x		Echo	opaque	4-40	(none)	x	x

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 1: Echo Option Summary

The Echo option value is generated by a server, and its content and structure are implementation specific. Different methods for generating Echo option values are outlined in Appendix A. Clients and intermediaries MUST treat an Echo option value as opaque and make no assumptions about its content or structure.

When receiving an Echo option in a request, the server MUST be able to verify that the Echo option value (a) was generated by the server or some other party that the server trusts, and (b) fulfills the freshness requirements of the application. Depending on the freshness requirements the server may verify exactly when the Echo option value was generated (time-based freshness) or verify that the Echo option was generated after a specific event (event-based freshness). As the request is bound to the Echo option value, the server can determine that the request is not older that the Echo option value.

When the Echo option is used with OSCORE [RFC8613] it MAY be an Inner or Outer option, and the Inner and Outer values are independent. OSCORE servers MUST only produce Inner Echo options unless they are merely testing for reachability of the client (the same as proxies may do). The Inner option is encrypted and integrity protected between the endpoints, whereas the Outer option is not protected by OSCORE and visible between the endpoints to the extent it is not protected by some other security protocol. E.g. in the case of DTLS hop-by-hop between the endpoints, the Outer option is visible to proxies along the path.

2.3. Echo Processing

The Echo option MAY be included in any request or response (see Section 2.4 for different applications).

The application decides under what conditions a CoAP request to a resource is required to be fresh. These conditions can for example include what resource is requested, the request method and other data in the request, and conditions in the environment such as the state of the server or the time of the day.

If a certain request is required to be fresh, the request does not contain a fresh Echo option value, and the server cannot verify the freshness of the request in some other way, the server MUST NOT process the request further and SHOULD send a 4.01 Unauthorized response with an Echo option. The server MAY include the same Echo option value in several different response messages and to different clients. Examples of this could be time-based freshness when several responses are sent closely after each other or event-based freshness with no event taking place between the responses.

The server may use request freshness provided by the Echo option to verify the aliveness of a client or to synchronize state. The server may also include the Echo option in a response to force a client to demonstrate reachability at its claimed network address. Note that the Echo option does not bind a request to any particular previous

response, but provides an indication that the client had access to the previous response at the time when it created the request.

Upon receiving a 4.01 Unauthorized response with the Echo option, the client SHOULD resend the original request with the addition of an Echo option with the received Echo option value. The client MAY send a different request compared to the original request. Upon receiving any other response with the Echo option, the client SHOULD echo the Echo option value in the next request to the server. The client MAY include the same Echo option value in several different requests to the server.

A client MUST only send Echo values to endpoints it received them from (where as defined in [RFC7252] Section 1.2, the security association is part of the endpoint). In OSCORE processing, that means sending Echo values from Outer options (or from non-OSCORE responses) back in Outer options, and those from Inner options in Inner options in the same security context.

Upon receiving a request with the Echo option, the server determines if the request is required to be fresh. If not, the Echo option MAY be ignored. If the request is required to be fresh and the server cannot verify the freshness of the request in some other way, the server MUST use the Echo option to verify that the request is fresh. If the server cannot verify that the request is fresh, the request is not processed further, and an error message MAY be sent. The error message SHOULD include a new Echo option.

One way for the server to verify freshness is that to bind the Echo value to a specific point in time and verify that the request is not older than a certain threshold T. The server can verify this by checking that (t1 - t0) < T, where t1 is the request receive time and t0 is the time when the Echo option value was generated. An example message flow is shown in Figure 2.

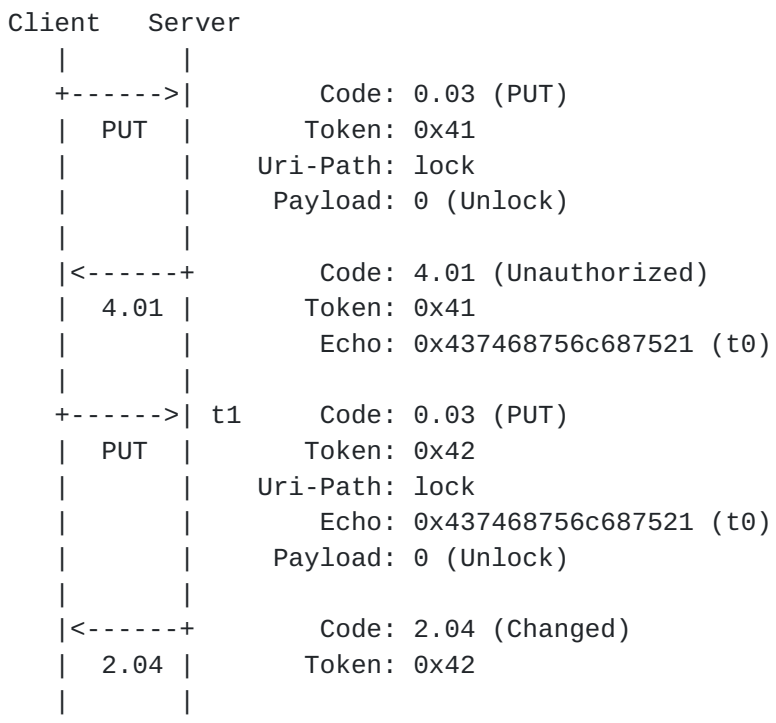

Figure 2: Example Message Flow for Time-Based Freshness

Another way for the server to verify freshness is to maintain a cache of values associated to events. The size of the cache is defined by the application. In the following we assume the cache size is 1, in which case freshness is defined as no new event has taken place. At each event a new value is written into the cache. The cache values MUST be different for all practical purposes. The server verifies freshness by checking that e0 equals e1, where e0 is the cached value when the Echo option value was generated, and e1 is the cached value at the reception of the request. An example message flow is shown in Figure 3.

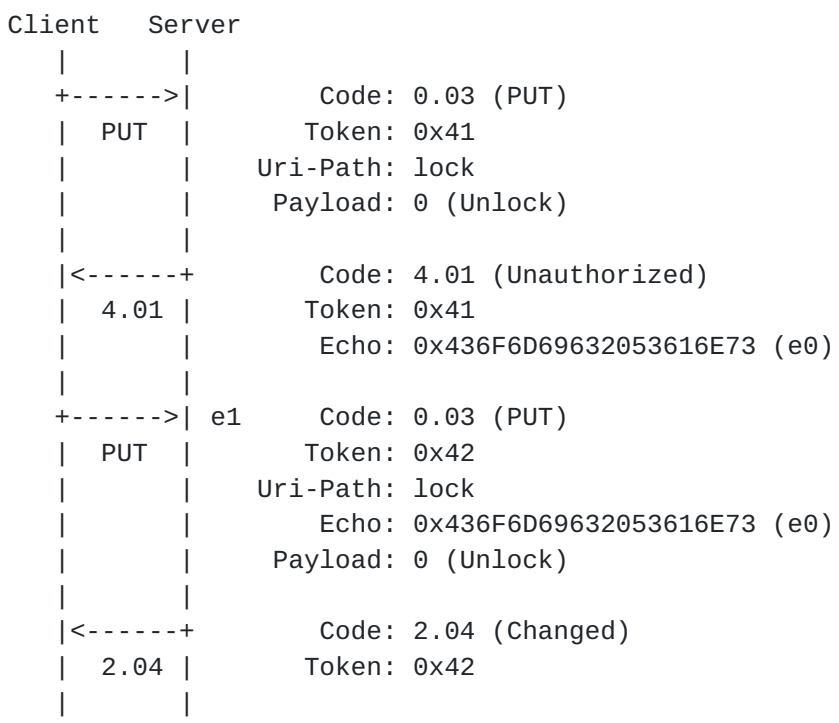

Figure 3: Example Message Flow for Event-Based Freshness

When used to serve freshness requirements (including client aliveness and state synchronizing), the Echo option value MUST be integrity protected between the intended endpoints, e.g. using DTLS, TLS, or an OSCORE Inner option ([RFC8613]). When used to demonstrate reachability at a claimed network address, the Echo option SHOULD contain the client's network address, but MAY be unprotected.

A CoAP-to-CoAP proxy MAY set an Echo option on responses, both on forwarded ones that had no Echo option or ones generated by the proxy (from cache or as an error). If it does so, it MUST remove the Echo option it recognizes as one generated by itself on follow-up requests. However, it MUST relay the Echo option of responses unmodified, and MUST relay the Echo option of requests it does not recognize as generated by itself unmodified.

The CoAP server side of CoAP-to-HTTP proxies MAY request freshness, especially if they have reason to assume that access may require it (e.g. because it is a PUT or POST); how this is determined is out of scope for this document. The CoAP client side of HTTP-to-CoAP proxies SHOULD respond to Echo challenges themselves if they know from the recent establishing of the connection that the HTTP request is fresh. Otherwise, they SHOULD respond with 503 Service Unavailable, Retry-After: 0 and terminate any underlying Keep-Alive connection. They MAY also use other mechanisms to establish freshness of the HTTP request that are not specified here.

2.4. Applications of the Echo Option

1. Actuation requests often require freshness guarantees to avoid accidental or malicious delayed actuator actions. In general, all non-safe methods (e.g. POST, PUT, DELETE) may require freshness guarantees for secure operation.

   * The same Echo value may be used for multiple actuation requests to the same server, as long as the total round-trip time since the Echo option value was generated is below the freshness threshold.

   * For actuator applications with low delay tolerance, to avoid additional round-trips for multiple requests in rapid sequence, the server may include the Echo option with a new value even in a successful response to a request, irrespectively of whether the request contained an Echo option or not. The client then uses the Echo option with the new value in the next actuation request, and the server compares the receive time accordingly.

2. A server may use the Echo option to synchronize properties (such as state or time) with a requesting client. A server MUST NOT synchronize a property with a client which is not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then server MUST NOT synchronize time with a client requesting access unless it is time authority for the server.

   * If a server reboots during operation it may need to synchronize state or time before continuing the interaction. For example, with OSCORE it is possible to reuse a partly persistently stored security context by synchronizing the Partial IV (sequence number) using the Echo option, see Section 7.5 of [RFC8613].

   * A device joining a CoAP group communication [RFC7390] protected with OSCORE [I-D.ietf-core-oscore-groupcomm] may be required to initially verify freshness and synchronize state or time with a client by using the Echo option in a unicast response to a multicast request. The client receiving the response with the Echo option includes the Echo option with the same value in a request, either in a unicast request to the responding server, or in a subsequent group request. In the latter case, the Echo option will be ignored except by the responding server.

3. A server that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks such as described in Section 11.3 of [RFC7252] (where an attacker would put a victim's address in the source address of a CoAP request). For this purpose, a server MAY ask a client to Echo its request to verify its source address. This needs to be done only once per peer and limits the range of potential victims from the general Internet to endpoints that have been previously in contact with the server. For this application, the Echo option can be used in messages that are not integrity protected, for example during discovery.

   * In the presence of a proxy, a server will not be able to distinguish different origin client endpoints. Following from the recommendation above, a proxy that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks. The proxy MAY use Echo to verify origin reachability as described in Section 2.3. The proxy MAY forward idempotent requests immediately to have a cached result available when the client's Echoed request arrives.

4. A server may want to use the request freshness provided by the Echo to verify the aliveness of a client. Note that in a deployment with hop-by-hop security and proxies, the server can only verify aliveness of the closest proxy.

3. Protecting Message Bodies using Request Tags

3.1. Fragmented Message Body Integrity

CoAP was designed to work over unreliable transports, such as UDP, and include a lightweight reliability feature to handle messages which are lost or arrive out of order. In order for a security protocol to support CoAP operations over unreliable transports, it must allow out-of-order delivery of messages using e.g. a sliding replay window such as described in Section 4.1.2.6 of DTLS ([RFC6347]).

The block-wise transfer mechanism [RFC7959] extends CoAP by defining the transfer of a large resource representation (CoAP message body) as a sequence of blocks (CoAP message payloads). The mechanism uses a pair of CoAP options, Block1 and Block2, pertaining to the request and response payload, respectively. The block-wise functionality does not support the detection of interchanged blocks between different message bodies to the same resource having the same block number. This remains true even when CoAP is used together with a security protocol such as DTLS or OSCORE, within the replay window

([I-D.mattsson-core-coap-actuators]), which is a vulnerability of CoAP when using RFC7959.

A straightforward mitigation of mixing up blocks from different messages is to use unique identifiers for different message bodies, which would provide equivalent protection to the case where the complete body fits into a single payload. The ETag option [RFC7252], set by the CoAP server, identifies a response body fragmented using the Block2 option.

3.2. The Request-Tag Option

This document defines the Request-Tag option for identifying request bodies, similar to ETag, but ephemeral and set by the CoAP client. The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct block-wise request operations on one resource from one client, addressing the issue described in Section 3.1. It enables the receiving server to reliably assemble request payloads (blocks) to their message bodies, and, if it chooses to support it, to reliably process simultaneous block-wise request operations on a single resource. The requests must be integrity protected if they should protect against interchange of blocks between different message bodies. The Request-Tag option is only used in requests that carry the Block1 option, and in Block2 requests following these.

In essence, it is an implementation of the "proxy-safe elective option" used just to "vary the cache key" as suggested in [RFC7959] Section 2.4.

3.2.1. Request-Tag Option Format

The Request-Tag option is not critical, is safe to forward, repeatable, and part of the cache key, see Figure 4, which extends Table 4 of [RFC7252]).

No.	C	U	N	R	Name	Format	Len.	Default	E	U
TBD				x	Request-Tag	opaque	0-8	(none)	x	x

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 4: Request-Tag Option Summary

Request-Tag, like the block options, is both a class E and a class U option in terms of OSCORE processing (see Section 4.1 of [RFC8613]):

The Request-Tag MAY be an Inner or Outer option. It influences the Inner or Outer block operation, respectively. The Inner and Outer values are therefore independent of each other. The Inner option is encrypted and integrity protected between client and server, and provides message body identification in case of end-to-end fragmentation of requests. The Outer option is visible to proxies and labels message bodies in case of hop-by-hop fragmentation of requests.

The Request-Tag option is only used in the request messages of block-wise operations.

The Request-Tag mechanism can be applied independently on the server and client sides of CoAP-to-CoAP proxies as are the block options, though given it is safe to forward, a proxy is free to just forward it when processing an operation. CoAP-to-HTTP proxies and HTTP-to-CoAP proxies can use Request-Tag on their CoAP sides; it is not applicable to HTTP requests.

3.3. Request-Tag Processing by Servers

The Request-Tag option does not require any particular processing on the server side outside of the processing already necessary for any unknown elective proxy-safe cache-key option: The option varies the properties that distinguish block-wise operations (which includes all options except elective NoCacheKey and except Block1/2), and thus the server can not treat messages with a different list of Request-Tag options as belonging to the same operation.

To keep utilizing the cache, a server (including proxies) MAY discard the Request-Tag option from an assembled block-wise request when consulting its cache, as the option relates to the operation-on-the-wire and not its semantics. For example, a FETCH request with the same body as an older one can be served from the cache if the older's Max-Age has not expired yet, even if the second operation uses a Request-Tag and the first did not. (This is similar to the situation about ETag in that it is formally part of the cache key, but implementations that are aware of its meaning can cache more efficiently, see [RFC7252] Section 5.4.2).

A server receiving a Request-Tag MUST treat it as opaque and make no assumptions about its content or structure.

Two messages carrying the same Request-Tag is a necessary but not sufficient condition for being part of the same operation. For one, a server may still treat them as independent messages when it sends 2.01/2.04 responses for every block. Also, a client that lost interest in an old operation but wants to start over can overwrite

the server's old state with a new initial (num=0) Block1 request and the same Request-Tag under some circumstances. Likewise, that results in the new message not being part of he old operation.

As it has always been, a server that can only serve a limited number of block-wise operations at the same time can delay the start of the operation by replying with 5.03 (Service unavailable) and a Max-Age indicating how long it expects the existing operation to go on, or it can forget about the state established with the older operation and respond with 4.08 (Request Entity Incomplete) to later blocks on the first operation.

3.4. Setting the Request-Tag

For each separate block-wise request operation, the client can choose a Request-Tag value, or choose not to set a Request-Tag. It needs to be set to the same value (or unset) in all messages belonging to the same operation, as otherwise they are treated as separate operations by the server.

Starting a request operation matchable to a previous operation and even using the same Request-Tag value is called request tag recycling. The absence of a Request-Tag option is viewed as a value distinct from all values with a single Request-Tag option set; starting a request operation matchable to a previous operation where neither has a Request-Tag option therefore constitutes request tag recycling just as well (also called "recycling the absent option").

Clients that use Request-Tag for a particular purpose (like in Section 3.5) MUST NOT recycle a request tag unless the first operation has concluded. What constitutes a concluded operation depends on that purpose, and is defined there.

When Block1 and Block2 are combined in an operation, the Request-Tag of the Block1 phase is set in the Block2 phase as well for otherwise the request would have a different set of options and would not be recognized any more.

Clients are encouraged to generate compact messages. This means sending messages without Request-Tag options whenever possible, and using short values when the absent option can not be recycled.

The Request-Tag options MAY be present in request messages that carry a Block2 option even if those messages are not part of a blockwise request operation (this is to allow the operation described in Section 3.5.3). The Request-Tag option MUST NOT be present in response messages, and MUST NOT be present if neither the Block1 nor the Block2 option is present.

3.5. Applications of the Request-Tag Option

3.5.1. Body Integrity Based on Payload Integrity

When a client fragments a request body into multiple message payloads, even if the individual messages are integrity protected, it is still possible for a man-in-the-middle to maliciously replace a later operation's blocks with an earlier operation's blocks (see Section 2.5 of [I-D.mattsson-core-coap-actuators]). Therefore, the integrity protection of each block does not extend to the operation's request body.

In order to gain that protection, use the Request-Tag mechanism as follows:

- The individual exchanges MUST be integrity protected end-to-end between client and server.

- The client MUST NOT recycle a request tag in a new operation unless the previous operation matchable to the new one has concluded.

  If any future security mechanisms allow a block-wise transfer to continue after an endpoint's details (like the IP address) have changed, then the client MUST consider messages sent to _any_ endpoint address within the new operation's security context.

- The client MUST NOT regard a block-wise request operation as concluded unless all of the messages the client previously sent in the operation have been confirmed by the message integrity protection mechanism, or are considered invalid by the server if replayed.

  Typically, in OSCORE, these confirmations can result either from the client receiving an OSCORE response message matching the request (an empty ACK is insufficient), or because the message's sequence number is old enough to be outside the server's receive window.

  In DTLS, this can only be confirmed if the request message was not retransmitted, and was responded to.

Authors of other documents (e.g. applications of [RFC8613]) are invited to mandate this behavior for clients that execute block-wise interactions over secured transports. In this way, the server can rely on a conforming client to set the Request-Tag option when required, and thereby conclude on the integrity of the assembled body.

Note that this mechanism is implicitly implemented when the security layer guarantees ordered delivery (e.g. CoAP over TLS [RFC8323]). This is because with each message, any earlier message can not be replayed any more, so the client never needs to set the Request-Tag option unless it wants to perform concurrent operations.

3.5.2. Multiple Concurrent Block-wise Operations

CoAP clients, especially CoAP proxies, may initiate a block-wise request operation to a resource, to which a previous one is already in progress, which the new request should not cancel. A CoAP proxy would be in such a situation when it forwards operations with the same cache-key options but possibly different payloads.

For those cases, Request-Tag is the proxy-safe elective option suggested in [RFC7959] Section 2.4 last paragraph.

When initializing a new block-wise operation, a client has to look at other active operations:

- If any of them is matchable to the new one, and the client neither wants to cancel the old one nor postpone the new one, it can pick a Request-Tag value (including the absent option) that is not in use by the other matchable operations for the new operation.

- Otherwise, it can start the new operation without setting the Request-Tag option on it.

3.5.3. Simplified Block-Wise Handling for Constrained Proxies

The Block options were defined to be unsafe to forward because a proxy that would forward blocks as plain messages would risk mixing up clients' requests.

The Request-Tag option provides a very simple way for a proxy to keep them separate: if it appends a Request-Tag that is particular to the requesting endpoint to all request carrying any Block option, it does not need to keep track of any further block state.

This is particularly useful to proxies that strive for stateless operation as described in [I-D.ietf-core-stateless] Section 3.1.

3.6. Rationale for the Option Properties

The Request-Tag option can be elective, because to servers unaware of the Request-Tag option, operations with differing request tags will not be matchable.

The Request-Tag option can be safe to forward but part of the cache key, because to proxies unaware of the Request-Tag option will consider operations with differing request tags unmatchable but can still forward them.

The Request-Tag option is repeatable because this easily allows stateless proxies to "chain" their origin address. They can perform the steps of Section 3.5.3 without the need to create an option value that is the concatenation of the received option and their own value, and can simply add a new Request-Tag option unconditionally.

In draft versions of this document, the Request-Tag option used to be critical and unsafe to forward. That design was based on an erroneous understanding of which blocks could be composed according to [RFC7959].

3.7. Rationale for Introducing the Option

An alternative that was considered to the Request-Tag option for coping with the problem of fragmented message body integrity (Section 3.5.1) was to update [RFC7959] to say that blocks could only be assembled if their fragments' order corresponded to the sequence numbers.

That approach would have been difficult to roll out reliably on DTLS where many implementations do not expose sequence numbers, and would still not prevent attacks like in [I-D.mattsson-core-coap-actuators] Section 2.5.2.

3.8. Block2 / ETag Processing

The same security properties as in Section 3.5.1 can be obtained for blockwise response operations. The threat model here is not an attacker (because the response is made sure to belong to the current request by the security layer), but blocks in the client's cache.

Rules stating that response body reassembly is conditional on matching ETag values are already in place from Section 2.4 of [RFC7959].

To gain equivalent protection to Section 3.5.1, a server MUST use the Block2 option in conjunction with the ETag option ([RFC7252], Section 5.10.6), and MUST NOT use the same ETag value for different representations of a resource.

4. Token Processing for Secure Request-Response Binding

4.1. Request-Response Binding

A fundamental requirement of secure REST operations is that the client can bind a response to a particular request. If this is not ensured, a client may erroneously associate the wrong response to a request. The wrong response may be an old response for the same resource or for a completely different resource (see e.g. Section 2.3 of [I-D.mattsson-core-coap-actuators]). For example, a request for the alarm status "GET /status" may be associated to a prior response "on", instead of the correct response "off".

In HTTPS, this type of binding is always assured by the ordered and reliable delivery as well as mandating that the server sends responses in the same order that the requests were received. The same is not true for CoAP where the server (or an attacker) can return responses in any order and where there can be any number of responses to a request (see e.g. [RFC7641]). In CoAP, concurrent requests are differentiated by their Token. Note that the CoAP Message ID cannot be used for this purpose since those are typically different for REST request and corresponding response in case of "separate response", see Section 2.2 of [RFC7252].

CoAP [RFC7252] does not treat Token as a cryptographically important value and does not give stricter guidelines than that the Tokens currently "in use" SHOULD (not SHALL) be unique. If used with a security protocol not providing bindings between requests and responses (e.g. DTLS and TLS) Token reuse may result in situations where a client matches a response to the wrong request. Note that mismatches can also happen for other reasons than a malicious attacker, e.g. delayed delivery or a server sending notifications to an uninterested client.

A straightforward mitigation is to mandate clients to not reuse Tokens until the traffic keys have been replaced. One easy way to accomplish this is to implement the Token as a counter starting at zero for each new or rekeyed secure connection.

4.2. Updated Token Processing Requirements for Clients

As described in Section 4.1, the client must be able to verify that a response corresponds to a particular request. This section updates the Token processing requirements for clients in [RFC7252] to always assure a cryptographically secure binding of responses to requests for secure REST operations like "coaps". The Token processing for servers is not updated. Token processing in Section 5.3.1 of [RFC7252] is updated by adding the following text:

When CoAP is used with a security protocol not providing bindings between requests and responses, the Tokens have cryptographic importance. The client MUST make sure that Tokens are not used in a way so that responses risk being associated with the wrong request. One easy way to accomplish this is to implement the Token (or part of the Token) as a sequence number starting at zero for each new or rekeyed secure connection, this approach SHOULD be followed.

5. Security Considerations

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of the Echo option. If no true random number generator is available, a truly random seed must be provided from an external source. As each pseudoranom number must only be used once, an implementation need to get a new truly random seed after reboot, or continously store state in nonvolatile memory, see ([RFC8613], Appendix B.1.1) for issues and solution approaches for writing to nonvolatile memory.

A single active Echo value with 64 (pseudo-)random bits gives the same theoretical security level as a 64-bit MAC (as used in e.g. AES_128_CCM_8). The Echo option value MUST contain 32 (pseudo-)random bits that are not predictable for any other party than the server, and SHOULD contain 64 (pseudo-)random bits. A server MAY use different security levels for different uses cases (client aliveness, request freshness, state synchronization, network address reachability, etc.).

The security provided by the Echo and Request-Tag options depends on the security protocol used. CoAP and HTTP proxies require (D)TLS to be terminated at the proxies. The proxies are therefore able to manipulate, inject, delete, or reorder options or packets. The security claims in such architectures only hold under the assumption that all intermediaries are fully trusted and have not been compromised.

Servers SHOULD use a monotonic clock to generate timestamps and compute round-trip times. Use of non-monotonic clocks is not secure as the server will accept expired Echo option values if the clock is moved backward. The server will also reject fresh Echo option values if the clock is moved forward. Non-monotonic clocks MAY be used as long as they have deviations that are acceptable given the freshness requirements. If the deviations from a monotonic clock are known, it may be possible to adjust the threshold accordingly.

An attacker may be able to affect the server's system time in various ways such as setting up a fake NTP server or broadcasting false time signals to radio-controlled clocks.

Servers MAY use the time since reboot measured in some unit of time. Servers MAY reset the timer at certain times and MAY generate a random offset applied to all timestamps. When resetting the timer, the server MUST reject all Echo values that was created before the reset.

Servers that use the List of Cached Random Values and Timestamps method described in Appendix A may be vulnerable to resource exhaustion attacks. One way to minimize state is to use the Integrity Protected Timestamp method described in Appendix A.

5.1. Token reuse

Reusing Tokens in a way so that responses are guaranteed to not be associated with the wrong request is not trivial as on-path attackers may block, delay, and reorder messages, requests may be sent to several servers, and servers may process requests in any order and send many responses to the same request. The use of a sequence number is therefore recommended when CoAP is used with a security protocol that does not providing bindings between requests and responses such as DTLS or TLS.

For a generic response to a confirmable request over DTLS, binding can only be claimed without out-of-band knowledge if

- the original request was never retransmitted,
- the response was piggybacked in an Acknowledgement message (as a confirmable or non-confirmable response may have been transmitted multiple times), and
- if observation was used, the same holds for the registration, all re-registrations, and the cancellation.

(In addition, for observations, any responses using that Token and a DTLS sequence number earlier than the cancellation Acknowledgement message need to be discarded. This is typically not supported in DTLS implementations.)

In some setups, Tokens can be reused without the above constraints, as a different component in the setup provides the associations:

- In CoAP over TLS, retransmissions are not handled by the CoAP layer and the replay window size is always exactly 1. When a client is sending TLS protected requests without Observe to a single server, the client can reuse a Token as soon as the previous response with that Token has been received.

- Requests whose responses are cryptographically bound to the requests (like in OSCORE) can reuse Tokens indefinitely.

In all other cases, a sequence number approach is RECOMMENDED as per Section 4.

Tokens that cannot be reused need to be handled appropriately. This could be solved by increasing the Token as soon as the currently used Token cannot be reused, or by keeping a list of all blacklisted Tokens.

When the Token (or part of the Token) contains a sequence number, the encoding of the sequence number has to be chosen in a way to avoid any collisions. This is especially true when the Token contains more information than just the sequence number, e.g. serialized state as in [I-D.ietf-core-stateless].

6. Privacy Considerations

Implementations SHOULD NOT put any privacy sensitive information in the Echo or Request-Tag option values. Unencrypted timestamps MAY reveal information about the server such as location or time since reboot, or that the server will accept expired certificates. Timestamps MAY be used if Echo is encrypted between the client and the server, e.g. in the case of DTLS without proxies or when using OSCORE with an Inner Echo option.

Like HTTP cookies, the Echo option could potentially be abused as a tracking mechanism to link to different requests to the same client. This is especially true for pre-emptive Echo values. Servers MUST NOT use the Echo option to correlate requests for other purposes than freshness and reachability. Clients only send Echo to the same from which they were received. Compared to HTTP, CoAP clients are often authenticated and non-mobile, and servers can therefore often correlate requests based on the security context, the client credentials, or the network address. When the Echo option increases a server's ability to correlate requests, clients MAY discard all pre-emptive Echo values.

7. IANA Considerations

This document adds the following option numbers to the "CoAP Option Numbers" registry defined by [RFC7252]:

Number	Name	Reference
TBD1	Echo	[[this document]]
TBD2	Request-Tag	[[this document]]

Figure 5: CoAP Option Numbers

8. References

8.1. Normative References

[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>.

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

[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>.

8.2. Informative References

[I-D.ietf-core-oscore-groupcomm] Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-05 (work in progress), July 2019.

[I-D.ietf-core-stateless] Hartke, K., "Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)", draft-ietf-core-stateless-03 (work in progress), October 2019.

[I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., Palombini, F., and C. Amsuess, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-06 (work in progress), September 2018.

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

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <https://www.rfc-editor.org/info/rfc7390>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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>.

Appendix A. Methods for Generating Echo Option Values

The content and structure of the Echo option value are implementation specific and determined by the server. Two simple mechanisms for time-based freshness are outlined in this section, the first is RECOMMENDED in general, and the second is RECOMMENDED in case the Echo option is encrypted between the client and the server.

Different mechanisms have different tradeoffs between the size of the Echo option value, the amount of server state, the amount of computation, and the security properties offered. A server MAY use different methods and security levels for different uses cases

(client aliveness, request freshness, state synchronization, network address reachability, etc.).

1. List of Cached Random Values and Timestamps. The Echo option value is a (pseudo-)random byte string. The server caches a list containing the random byte strings and their transmission times. Assuming 72-bit random values and 32-bit timestamps, the size of the Echo option value is 9 bytes and the amount of server state is 13n bytes, where n is the number of active Echo Option values. The security against an attacker guessing echo values is given by s = bit length of r - log2(n). The length of r and the maximum allowed n should be set so that the security level is harmonized with other parts of the deployment, e.g., s >= 64. If the server loses time continuity, e.g. due to reboot, the entries in the old list MUST be deleted.

```
Echo option value: random value r
Server State: random value r, timestamp t0
```

2. Integrity Protected Timestamp. The Echo option value is an integrity protected timestamp. The timestamp can have different resolution and range. A 32-bit timestamp can e.g. give a resolution of 1 second with a range of 136 years. The (pseudo-)random secret key is generated by the server and not shared with any other party. The use of truncated HMAC-SHA-256 is RECOMMENDED. With a 32-bit timestamp and a 64-bit MAC, the size of the Echo option value is 12 bytes and the Server state is small and constant. The security against an attacker guessing echo values is given by the MAC length. If the server loses time continuity, e.g. due to reboot, the old key MUST be deleted and replaced by a new random secret key. Note that the privacy considerations in Section 6 may apply to the timestamp. A server MAY want to encrypt its timestamps, and, depending on the choice of encryption algorithms, this may require a nonce to be included in the Echo option value.

```
Echo option value: timestamp t0, MAC(k, t0)
Server State: secret key k
```

Other mechanisms complying with the security and privacy considerations may be used. The use of encrypted timestamps in the Echo option increases security, but typically requires an IV to be included in the Echo option value, which adds overhead and makes the specification of such a mechanism slightly more complicated than the two mechanisms specified here.

Appendix B. Request-Tag Message Size Impact

In absence of concurrent operations, the Request-Tag mechanism for body integrity (Section 3.5.1) incurs no overhead if no messages are lost (more precisely: in OSCORE, if no operations are aborted due to repeated transmission failure; in DTLS, if no packages are lost), or when block-wise request operations happen rarely (in OSCORE, if there is always only one request block-wise operation in the replay window).

In those situations, no message has any Request-Tag option set, and that can be recycled indefinitely.

When the absence of a Request-Tag option can not be recycled any more within a security context, the messages with a present but empty Request-Tag option can be used (1 Byte overhead), and when that is used-up, 256 values from one byte long options (2 Bytes overhead) are available.

In situations where those overheads are unacceptable (e.g. because the payloads are known to be at a fragmentation threshold), the absent Request-Tag value can be made usable again:

- In DTLS, a new session can be established.

- In OSCORE, the sequence number can be artificially increased so that all lost messages are outside of the replay window by the time the first request of the new operation gets processed, and all earlier operations can therefore be regarded as concluded.

Appendix C. Change Log

[The editor is asked to remove this section before publication.]

- Changes since draft-ietf-core-echo-request-tag-07 (largely addressing Francesca's review):

  * Request tag: Explicitly limit "MUST NOT recycle" requirement to particular applications

  * Token reuse: upper-case RECOMMEND sequence number approach

  * Structure: Move per-topic introductions to respective chapters (this avoids long jumps by the reader)

  * Structure: Group Block2 / ETag section inside new fragmentation (formerly Request-Tag) section

    * More precise references into other documents

    * "concurrent operations": Emphasise that all here only matters between endpoint pairs

    * Freshness: Generalize wording away from time-based freshness

    * Echo: Emphasise that no binding between any particular pair of responses and requests is established

    * Echo: Add event-based example

    * Echo: Clarify when protection is needed

    * Request tag: Enhance wording around "not sufficient condition"

    * Request tag: Explicitly state when a tag needs to be set

    * Request tag: Clarification about permissibility of leaving the option absent

    * Security considerations: wall clock time -> system time (and remove inaccurate explanations)

    * Token reuse: describe blacklisting in a more implementation-independent way

o Changes since draft-ietf-core-echo-request-tag-06:

    * Removed visible comment that should not be visible in Token reuse considerations.

o Changes since draft-ietf-core-echo-request-tag-05:

    * Add privacy considerations on cookie-style use of Echo values

    * Add security considerations for token reuse

    * Add note in security considerations on use of nonvolatile memory when dealing with pseudorandom numbers

    * Appendix on echo generation: add a few words on up- and downsides of the encrypted timestamp alternative

    * Clarifications around Outer Echo:

        + Could be generated by the origin server to prove network reachability (but for most applications it MUST be inner)

    + Could be generated by intermediaries

    + Is answered by the client to the endpoint from which it received it (ie. Outer if received as Outer)

  * Clarification that a server can send Echo preemtively

  * Refer to stateless to explain what "more information than just the sequence number" could be

  * Remove explanations around 0.00 empty messags

  * Rewordings:

    + the attack: from "forging" to "guessing"

    + "freshness tokens" to "freshness indicators" (to avoid confusion with the Token)

  * Editorial fixes:

    + Abstract and introduction mention what is updated in RFC7252

    + Reference updates

    + Capitalization, spelling, terms from other documents

- Changes since draft-ietf-core-echo-request-tag-04:

  * Editorial fixes

    + Moved paragraph on collision-free encoding of data in the Token to Security Considerations and rephrased it

    + "easiest" -> "one easy"

- Changes since draft-ietf-core-echo-request-tag-03:

  * Mention Token processing changes in title

  * Abstract reworded

  * Clarify updates to Token processing

  * Describe security levels from Echo length

  * Allow non-monotonic clocks under certain conditions for freshness

    * Simplify freshness expressions

    * Describe when a Request-Tag can be set

    * Add note on application-level freshness mechanisms

    * Minor editorial changes

- Changes since draft-ietf-core-echo-request-tag-02:

    * Define "freshness"

    * Note limitations of "aliveness"

    * Clarify proxy and OSCORE handling in presence of "echo"

    * Clarify when Echo values may be reused

    * Update security considerations

    * Various minor clarifications

    * Minor editorial changes

- Major changes since draft-ietf-core-echo-request-tag-01:

    * Follow-up changes after the "relying on block-wise" change in -01:

        + Simplify the description of Request-Tag and matchability

        + Do not update RFC7959 any more

    * Make Request-Tag repeatable.

    * Add rationale on not relying purely on sequence numbers.

- Major changes since draft-ietf-core-echo-request-tag-00:

    * Reworded the Echo section.

    * Added rules for Token processing.

    * Added security considerations.

    * Added actual IANA section.

  * Made Request-Tag optional and safe-to-forward, relying on block-wise to treat it as part of the cache-key

  * Dropped use case about OSCORE Outer-block-wise (the case went away when its Partial IV was moved into the Object-Security option)

- Major changes since draft-amsuess-core-repeat-request-tag-00:

  * The option used for establishing freshness was renamed from "Repeat" to "Echo" to reduce confusion about repeatable options.

  * The response code that goes with Echo was changed from 4.03 to 4.01 because the client needs to provide better credentials.

  * The interaction between the new option and (cross) proxies is now covered.

  * Two messages being "Request-Tag matchable" was introduced to replace the older concept of having a request tag value with its slightly awkward equivalence definition.

Acknowledgments

The authors want to thank Carsten Bormann, Francesca Palombini, and Jim Schaad for providing valuable input to the draft.

Authors' Addresses

Christian Amsuess

Email: christian@amsuess.com

John Preuss Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com