

CoRE Working Group  
Internet-Draft  
Updates: 7959 (if approved)  
Intended status: Standards Track  
Expires: May 3, 2018

C. Amsuess  
Energy Harvesting Solutions  
J. Mattsson  
G. Selander  
Ericsson AB  
October 30, 2017

Echo and Request-Tag  
draft-ietf-core-echo-request-tag-00

Abstract

This document defines two optional extensions to the Constrained Application Protocol (CoAP): the Echo option and the Request-Tag option. Each of these options when integrity protected, such as with DTLS or OSCORE, protects against certain attacks on CoAP message exchanges.

The Echo option enables a CoAP server to verify the freshness of a request by requiring the CoAP client to make another request and include a server-provided challenge. The Request-Tag option allows the CoAP server to match message fragments belonging to the same request message, fragmented using the CoAP Block-Wise Transfer mechanism. This document also specifies additional processing requirements on Block1 and Block2 options.

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 3, 2018.

## Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Request Freshness . . . . .	3
1.2. Fragmented Message Body Integrity . . . . .	3
1.3. Terminology . . . . .	4
2. The Echo Option . . . . .	5
2.1. Option Format . . . . .	5
2.2. Echo Processing . . . . .	5
2.3. Applications . . . . .	7
3. The Request-Tag Option . . . . .	8
3.1. Option Format . . . . .	9
3.2. Request-Tag Processing . . . . .	10
3.3. Applications . . . . .	11
3.3.1. Body Integrity Based on Payload Integrity . . . . .	11
3.3.2. Multiple Concurrent Blockwise Operations . . . . .	12
4. Block2 / ETag Processing . . . . .	13
5. IANA Considerations . . . . .	13
6. Security Considerations . . . . .	13
7. References . . . . .	13
7.1. Normative References . . . . .	13
7.2. Informative References . . . . .	14
Appendix A. Performance Impact When Using the Echo Option . . . .	14
Appendix B. Request-Tag Message Size Impact . . . . .	15
Appendix C. Change Log . . . . .	16
Authors' Addresses . . . . .	16

## 1. Introduction

The initial CoAP suite of specifications ([RFC7252], [RFC7641], [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 two server-oriented CoAP options, the Echo option and the Request-Tag option, addressing the security features request freshness and fragmented message body integrity, respectively. 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 in a message, e.g. using DTLS ([RFC6347]) or OSCORE ([I-D.ietf-core-object-security]), provide those security features. The Request-Tag option and also the ETag option are mandatory to use with Block1 and Block2, respectively, to secure blockwise operations with multiple representations of a particular resource as is specified in this document.

Additional applications of the options are introduced. For example, Echo can be used to mitigate amplification attacks.

### 1.1. Request Freshness

A CoAP server receiving a request may not be 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 incriminating in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks are possible to mitigate by establishing fresh session keys (e.g. performing the DTLS handshake) for each actuation, but in general this is not a solution suitable for constrained environments.

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. The Echo option, defined in this document, specifies such a mechanism which thereby enables the CoAP server to verify the freshness of a request. 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.

### 1.2. 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 blockwise functionality does not support the detection of interchanged blocks between different message bodies to the same endpoint 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.amsuess-core-request-tag]), 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. This document defines the Request-Tag option for identifying the request body fragmented using the Block1 option, similar to ETag, but ephemeral and set by the CoAP client.

### 1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Unless otherwise specified, the terms "client" and "server" refers to "CoAP client" and "CoAP server", respectively, as defined in [RFC7252].

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 "blockwise operation". A blockwise operation which is fragmenting the request body is called a "blockwise request operation". A blockwise operation which is fragmenting the response body is called a "blockwise response operation".

Two blockwise operations between the same endpoint pair on the same resource 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 blockwise request operations are impossible with the options of [RFC7959] because the

second operation's block overwrites any state of the first exchange.).

The Echo and Request-Tag options are defined in this document. The concept of two messages being "Request-Tag-matchable" is defined in Section 3.1.

## 2. The Echo Option

The Echo option is a server-driven challenge-response mechanism for CoAP. The Echo option value is a challenge from the server to the client included in a CoAP response and echoed in a CoAP request.

### 2.1. Option Format

The Echo Option is elective, safe-to-forward, not part of the cache-key, and not repeatable, see Figure 1.

No.	C	U	N	R	Name	Format	Length	Default	E
TBD					Echo	opaque	8-40	(none)	x

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

Figure 1: Echo Option Summary

The value of the Echo option MUST be a (pseudo-)random bit string of a length of at least 64 bits. A new (pseudo-)random bit string MUST be generated by the server for each use of the Echo option.

### 2.2. Echo Processing

It is important to identify 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.

A server MAY include the Echo option in a response. The Echo option MUST NOT be used with empty CoAP requests (i.e. Code=0.00). If the server receives a request which has freshness requirements, and the request does not contain the Echo option, the server SHOULD send a 4.01 Unauthorized response with a Echo option. The server SHOULD

cache the transmitted Echo option value and the response transmit time (here denoted  $t_0$ ).

Upon receiving a response with the Echo option within the EXCHANGE\_LIFETIME ([RFC7252]) of the original request, the client SHOULD echo the Echo option with the same value in a new request to the server. Upon receiving a 4.01 Unauthorized response with the Echo option in response to a request within the EXCHANGE\_LIFETIME of the original request, the client SHOULD resend the original request. The client MAY send a different request compared to the original request.

If the server receives a request which has freshness requirements, and the request contains the Echo option, the server MUST verify that the option value equals a cached value; otherwise the request is not processed further. The server MUST calculate the round-trip time  $RTT = (t_1 - t_0)$ , where  $t_1$  is the request receive time. The server MUST only accept requests with a round-trip time below a certain threshold  $T$ , i.e.  $RTT < T$ , otherwise the request is not processed further, and an error message MAY be sent. The threshold  $T$  is application specific, its value depends e.g. on the freshness requirements of the request. An example message flow is illustrated in Figure 2.

When used to serve freshness requirements, CoAP messages containing the Echo option MUST be integrity protected, e.g. using DTLS or OSCORE ([I-D.ietf-core-object-security]).

If the server loses time synchronization, e.g. due to reboot, it MUST delete all cached Echo option values and response transmission times.

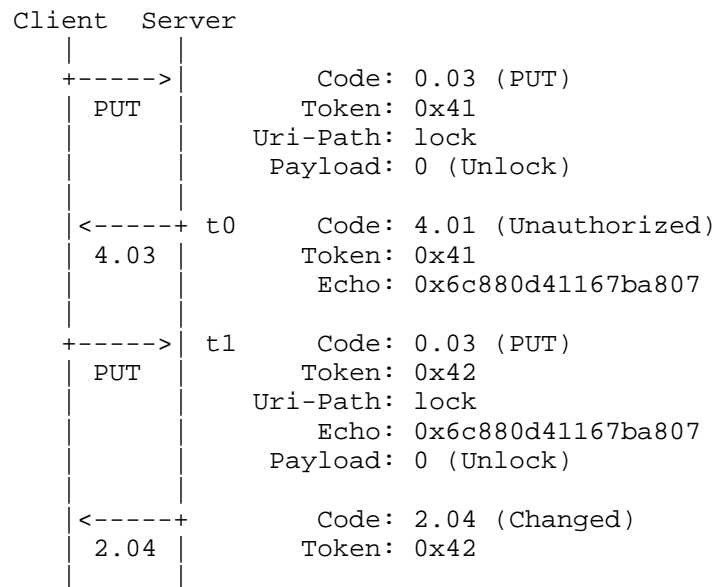


Figure 2: Echo option message flow

Constrained server implementations can use the mechanisms outlined in Appendix A to minimize the memory impact of having many unanswered Echo responses.

CoAP-CoAP proxies MUST relay the Echo option unmodified, and SHOULD NOT cache responses when a Echo option is present in request or response for more than the exchange. CoAP-HTTP proxies MAY request freshness, especially if they have reason to assume that access may require it (eg. because it is a PUT or POST); how this is determined is out of scope for this document. HTTP-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. It MAY also use other mechanisms to establish freshness of the HTTP request that are not specified here.

### 2.3. Applications

1. Actuation requests often require freshness guarantees to avoid accidental or malicious delayed actuator actions.
2. To avoid additional roundtrips for applications with multiple actuator requests in rapid sequence between the same client and

server, the server may use the Echo option (with a new value) in response to a request containing the Echo option. The client then uses the Echo option with the new value in the next actuation request, and the server compares the receive time accordingly.

3. If a server reboots during operation it may need to synchronize state with requesting clients before continuing the interaction. For example, with OSCORE it is possible to reuse a persistently stored security context by synchronizing the Partial IV (sequence number) using the Echo option.
4. When a device joins a multicast/broadcast group the device may need to synchronize state or time with the sender to ensure that the received message is fresh. By synchronizing time with the broadcaster, time can be used for synchronizing subsequent broadcast messages. A server MUST NOT synchronize state or time with clients which are not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then the client MUST NOT set the time of the server.
5. 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, the 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.

### 3. The Request-Tag Option

The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct blockwise request operations on one resource from one client. 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 blockwise request operations on a single resource. The requests must be integrity protected in order to protect against interchange of blocks between different message bodies.



### 3.1. Option Format

The Request-Tag option has the same properties as the Block1 option: it is critical, unsafe, not part of the cache-key, and not repeatable, see Figure 3.

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

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

Figure 3: Request-Tag Option Summary

[Note to RFC editor: If this document is not released together with OSCORE but before it, the following paragraph and the "E" column above need to move into OSCORE.]

Request-Tag, like the Block1 option, is a special class E option in terms of OSCORE processing (see Section 4.3.1.2 of [I-D.ietf-core-object-security]): The Request-Tag MAY be an inner or outer option. 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 request messages, and only in conjunction with the Block1 option.

Two messages are defined to be Request-Tag-matchable if and only if they are sent from and to the same end points (including security associations), and target the same URI, and if either neither carries a Request-Tag option, or both carry exactly one Request-Tag option and the option values are of same length and content.

The Request-Tag mechanism is applied independently on the server and client sides of CoAP-CoAP proxies. CoAP-HTTP proxies and HTTP-CoAP proxies can use Request-Tag on their CoAP sides; it is not applicable to HTTP requests.

For each separate blockwise request operation, the client can choose a Request-Tag value, or choose not to set a Request-Tag. Creating a new request operation whose messages are Request-Tag-matchable to a

previous operation is called request tag recycling. Clients MUST NOT recycle a request tag unless the first operation has concluded. What constitutes a concluded operation depends on the application, and is outlined individually in Section 3.3.

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.

### 3.2. Request-Tag Processing

A server MUST NOT act on any two blocks in the same blockwise request operation that are not Request-Tag-matchable. This rule applies independent of whether the request actually carries a Request-Tag option (in this case, the request can only be acted on together with other messages not carrying the option, as per matchability definition).

As not all messages from the same source can be combined any more, a block not matchable to the first Block1 cannot overwrite context kept for an operation under a different tag (cf. [RFC7959] Section 2.5). The server is still under no obligation to keep state of more than one transaction. When an operation is in progress and a second one cannot be served at the same time, the server MUST respond to the second request with a 5.03 (Service Unavailable) response code and SHOULD indicate the time it is willing to wait for additional blocks in the first operation using the Max-Age option, as specified in Section 5.9.3.4 of [RFC7252].

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

Two messages being Request-Tag-matchable is a necessary but not sufficient condition for being part of the same operation. They can still be treated as independent messages by the server (e.g. when it sends 2.01/2.04 responses for every block), or initiate a new operation (overwriting kept context) when the later message carries Block1 number 0.

If a request that uses Request-Tag is rejected with 4.02 Bad Option, the client MAY retry the operation without it, but then it MUST serialize all operations that affect the same resource. Security requirements can forbid dropping the use of Request-Tag mechanism.

### 3.3. Applications

#### 3.3.1. 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 later operation's blocks with earlier operation's blocks (see Section 3.2 of [I-D.amsuess-core-request-tag]). 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:

- o The message payloads MUST be integrity protected end-to-end between client and server.
- o The client MUST NOT recycle a request tag unless the previous blockwise request operation that used matchable Request-Tags has concluded.
- o The client MUST NOT regard a blockwise 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.

- o The client MUST NOT fall back to not using the Request-Tag mechanisms when receiving a 4.02 Bad Option response.

Authors of other documents (e.g. [I-D.ietf-core-object-security]) are invited to mandate this behavior for clients that execute blockwise 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 [I-D.tschofenig-core-coap-tcp-tls]). This is because with each message, any earlier operation can be regarded as concluded by the client, so it never needs to set the Request-Tag option unless it wants to perform concurrent operations.

### 3.3.2. Multiple Concurrent Blockwise Operations

CoAP clients, especially CoAP proxies, may initiate a blockwise request operation to a resource, to which a previous one is already in progress, and which the new request should not cancel. One example is when a CoAP proxy fragments an OSCORE messages using blockwise (so-called "outer" blockwise, see Section 4.3.1. of [I-D.ietf-core-object-security])), where the Uri-Path is hidden inside the encrypted message, and all the proxy sees is the server's "/" path.

When a client fragments a message as part of a blockwise request operation, it can do so without a Request-Tag option set. For this application, an operation can be regarded as concluded when a final Block1 option has been sent and acknowledged, or when the client chose not to continue with the operation (e.g. by user choice, or in the case of a proxy when it decides not to take any further messages in the operation due to a timeout). When another concurrent blockwise request operation is made (i.e. before the operation is concluded), the client can not recycle the request tag, and has to pick a new one. The possible outcomes are:

- o The server responds with a successful code.

The concurrent blockwise operations can then continue.

- o The server responds 4.02 Bad Option.

This can indicate that the server does not support Request-Tag. The client should wait for the first operation to conclude, and then try the same request without a Request-Tag option.

- o The server responds 5.03 Service Unavailable with a Max-Age option to indicate when it is likely to be available again.

This can indicate that the server supports Request-Tag, but still is not prepared to handle concurrent requests. The client should wait for as long as the response is valid, and then retry the operation, which may not need to carry a Request-Tag option by then any more.

In the cases where a CoAP proxy receives an error code, it can indicate the anticipated delay by sending a 5.03 Service Unavailable response itself. Note that this behavior is no different from what a CoAP proxy would need to do were it unaware of the Request-Tag option.

#### 4. Block2 / ETag Processing

The same security properties as in Section 3.3.1 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.

Analogous rules to Section 3.2 are already in place for assembling a response body in Section 2.4 of [RFC7959].

To gain equivalent protection to Section 3.3.1, 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.

#### 5. IANA Considerations

[TBD: Fill out the option templates for Echo and Request-Tag]

#### 6. Security Considerations

Servers that store a Echo challenge per client can be attacked for resource exhaustion, and should consider minimizing the state kept per client, e.g. using a mechanism as described in Appendix A.

#### 7. References

##### 7.1. 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>>.

## 7.2. Informative References

- [I-D.amsuess-core-request-tag]  
Amsuess, C., "Request-Tag option", draft-amsuess-core-request-tag-00 (work in progress), March 2017.
- [I-D.ietf-core-object-security]  
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-06 (work in progress), October 2017.
- [I-D.mattsson-core-coap-actuators]  
Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-02 (work in progress), November 2016.
- [I-D.tschofenig-core-coap-tcp-tls]  
Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-tschofenig-core-coap-tcp-tls-05 (work in progress), November 2015.
- [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>>.
- [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>>.

## Appendix A. Performance Impact When Using the Echo Option

The Echo option requires the server to keep some state in order to later verify the echoed request.

Instead of caching Echo option values and response transmission times, the server MAY use the encryption of the response transmit time  $t_0$  as Echo option value. Such a scheme needs to ensure that the server can detect a replay of a previous encrypted response transmit time.

For example, the server MAY encrypt  $t_0$  with AES-CCM-128-64-64 using a (pseudo-)random secret key  $k$  generated and cached by the server. A unique IV MUST be used with each encryption, e.g. using a sequence number. If the server loses time synchronization, e.g. due to reboot, then  $k$  MUST be deleted and replaced by a new random secret key. When using encrypted response transmit times, the Echo processing is modified in the following way: The verification of cached option value in the server processing is replaced by the verification of the integrity of the encrypted option value using the cached key and IV (e.g. sequence number).

The two methods - (a) the list of cached values, and (b) the encryption of transmit time - have different impact on the implementation:

- o size of cached data (list of cached values vs. key and IV)
- o size of message (typically larger with encrypted time)
- o computation (encryption + decryption vs. generation new nonce + cache + lookup)

In general, the encryption of transmission times is most useful if the number of concurrent requests is high.

A hybrid scheme is also possible: the first Echo option values are cached, and if the number of concurrent requests reach a certain threshold, then encrypted times are used until there is space for storing new values in the list. In that case, the server may need to make both verifications - either that the Echo value is in the list, or that it verifies in decryption - and in either case that the transmission time is valid.

#### Appendix B. Request-Tag Message Size Impact

In absence of concurrent operations, the Request-Tag mechanism for body integrity (Section 3.3.1) 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 blockwise request operations happen rarely (in OSCORE, if only one request operation with losses within 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:

- o In DTLS, a new session can be established.
- o 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. ]

- o 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.

#### Authors' Addresses

Christian Amsuess  
Energy Harvesting Solutions  
  
Email: c.amsuess@energyharvesting.at

John Mattsson  
Ericsson AB  
  
Email: john.mattsson@ericsson.com



Goeran Selander  
Ericsson AB

Email: [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)