CoRE Working Group
Internet-Draft
Updates: 7252, 7959 (if approved)
Intended status: Standards Track
Expires: September 6, 2018

C. Amsuess
J. Mattsson
G. Selander
Ericsson AB

March 05, 2018

Echo and Request-Tag
draft-ietf-core-echo-request-tag-01

Abstract

This document specifies several security enhancements to the Constrained Application Protocol (CoAP). Two optional extensions are defined: the Echo option and the Request-Tag option. Each of these options provide additional features to CoAP and protects against certain attacks. The document also updates the processing requirements on the Block options and the Token. The updated Token processing ensures secure binding of responses to requests.

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 several security enhancements to the Constrained Application Protocol (CoAP).

This document specifies two server-oriented CoAP options, the Echo option and the Request-Tag option, mainly addressing the security features request freshness and fragmented message body integrity,

respectively. The Echo option enables a CoAP server to verify the freshness of a request, verify the aliveness of a client, synchronize state, or force a client to demonstrate reachability at its apparent 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 blockwise 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 ([RFC5246]), or OSCORE ([I-D.ietf-core-object-security]), provide the additional security features.

The document also updates the processing requirements on the Block1 option, the Block2 option, and the Token. The updated blockwise processing secure blockwise operations with multiple representations of a particular resource. The updated Token processing ensures secure binding of responses to requests.

1.1. 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 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 and to verify aliveness of the client.

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 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. 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. Request-Response Binding

A fundamental requirement of secure REST operations is that the client can bind a response to a particular request. In HTTPS this is 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 can return responses in any order. Concurrent requests are instead differentiated by their Token. Unfortunately, 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 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 (see e.g. Section 2.3 of [I-D.mattsson-core-coap-actuators]). 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 never reuse tokens until the traffic keys have been replaced. As there may be

any number of responses to a request (see e.g. [RFC7641]), the easiest way to accomplish this is to implement the token as a counter and never reuse any tokens at all. This document updates the Token processing in [RFC7252] to always assure a cryptographically secure binding of responses to requests.

1.4. 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 one or more 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			x		Echo	opaque	4-40	(none)	x

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

Figure 1: Echo 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.]

The Echo option value is generated by the 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 was generated by the server as well as the point in time when the Echo option value was generated.

2.2. Echo Processing

The Echo option MAY be included in any request or response (see Section 2.3 for different applications), but 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, 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 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.

The server may also include the Echo option in a response to verify the aliveness of a client, to synchronize state, or to force a client to demonstrate reachability at their apparent network address.

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 a next request to the server. The client MAY include the same Echo option value in several different requests to the server.

Upon receiving a request with the Echo option, the server determines if the request has freshness requirement. If the request does not have freshness requirements, the Echo option MAY be ignored. If the request has freshness requirements and the server cannot verify the freshness of the request in some other way, the server MUST verify that the Echo option value was generated by the server; otherwise the request is not processed further. The server MUST then calculate the round-trip time RTT = (t1 - t0), where t1 is the request receive time and t0 is the transmit time of the response that included the specific Echo option value. 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.

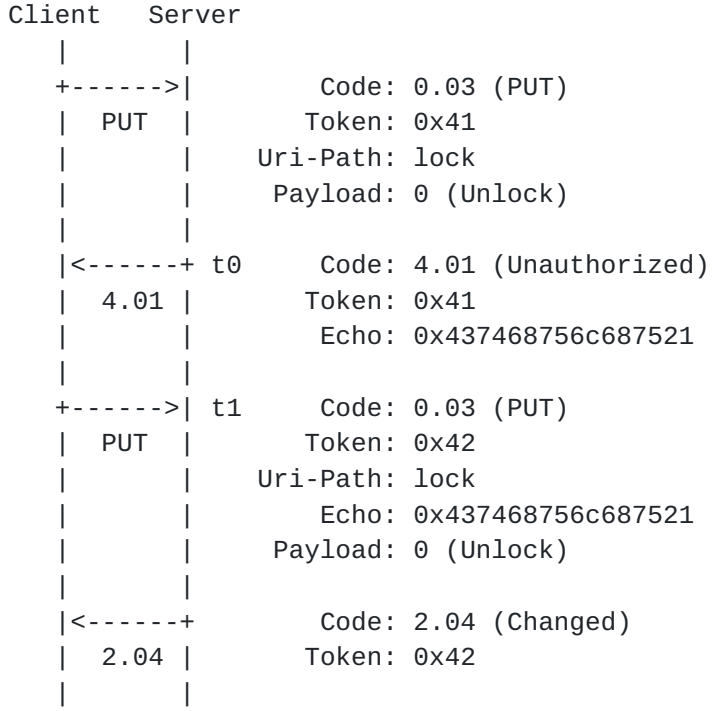

Figure 2: Example Echo Option Message Flow

When used to serve freshness requirements (including client aliveness and state synchronizing), CoAP requests containing the Echo option MUST be integrity protected, e.g. using DTLS, TLS, or OSCORE ([I-D.ietf-core-object-security]). When used to demonstrate reachability at their apparent network address, the Echo option MAY be used without protection.

Note that the server does not have to synchronize the time used for the Echo timestamps with any other party. If the server loses time synchronization, e.g. due to reboot, it MUST reject all Echo values that was created before time synchronization was lost.

CoAP-CoAP proxies MUST relay the Echo option unmodified. The CoAP server side of CoAP-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-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.

2.3. Applications

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.

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 partly 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.

6. A server may want to verify the aliveness of a client by responding with an Echo option.

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 is not critical, safe to forward, and part of the cache key as illustrated in Figure 3.

No.	C	U	N	R	Name	Format	Length	Default	E
TBD					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 block options, 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 the request messages of blockwise request operations.

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 (precisely: target the same endpoint and cache-key except for cache-key options that are related to blockwise), 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 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-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 (if not, 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 SHOULD respond to the second request with a 5.03 (Service Unavailable) response code and 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]. (Alternatively, the server can cancel the original operation, especially if it is already likely to time out. Cancelling it unconditionally is the behavior that could be expected of a Request-Tag unaware server.)

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.

Note that RFC 7959 already implies that the cache key is the element that binds exchanges together to operations (together with the request's source endpoint), but is not explicit about it; therefore, the above rules are spelt out here.

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 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 unless the previous blockwise request operation that used matchable Request-Tags has concluded.

- 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.

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 [RFC8323]). 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. A CoAP proxy would be in such a situation when it forwards operations with the same cache-key options but possibly different payloads.

When a client fragments an initial 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:

- The server responds with a successful code.

  The second concurrent blockwise operations can then continue.

  The first operation might have been cancelled by that (typical of servers that only support a single blockwise operation), in which case its resumption will result in a 4.08 Request Entity Incomplete error.

- 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 this, the proxy can indicate the anticipated delay by sending a 5.03 Service Unavailable response itself.

Note that a correctly implemented Request-Tag unaware proxy in the same situation would need to make a choice to either send a 5.03 with Max-Age by itself (holding off the second operation), or to commence the second operation and reject any further requests on the first operation with 4.08 Request Entity Incompelte errors by itself without forwarding them.

3.4. Rationale for the option properties

The Request-Tag option used to be critical and unsafe to forward in earlier revisions of this draft.

Given that supporting it will be mandated for where it is used for its security properties, the choice of whether it is mandatory or safe to forward can be made as required for the multiple concurrent operations use case. For those cases, Request-Tag is the proxy-safe elective option suggested in [RFC7959] Section 2.4 last paragraph.

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. Token Processing

This section updates the Token processing in Section 5.3.1 of [RFC7252] by adding the following text:

When CoAP is used with a security protocol not providing bindings between requests and responses, the client MUST NOT reuse tokens until the traffic keys have been replaced. The easiest way to accomplish this is to implement the Token as a counter, this approach SHOULD be followed.

6. IANA Considerations

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

Number	Name	Reference
TBD1	Echo	[RFC XXXX]
TBD2	Request-Tag	[RFC XXXX]

Figure 4: CoAP Option Numbers

7. Security Considerations

Servers 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 its wall clock time or location. Servers MUST use a monotonic clock to generate timestamps and compute round-trip times. Servers SHOULD NOT use wall clock time for

timestamps, as wall clock time is not monotonic, may reveal that the server will accept expired certificates, or reveal the server's location. 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. An attacker may be able to affect the server's wall clock time in various ways such as setting up a fake NTP server or broadcasting false time signals to radio-controlled clocks. Servers SHOULD use the time since reboot measured in some unit of time. Servers MAY reset the timer periodically even when not rebooting.

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.

An Echo value with 64 (pseudo-)random bits gives the same theoretical security level against forgeries as a 64-bit MAC (as used in e.g. AES_128_CCM_8). In practice, forgery of an Echo option value is much harder as an attacker must also forge the MAC in the security protocol. 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 that use the List of Cached Random Values and Timestamps method described in Appendix A may be vulnerable to resource exhaustion attacks. On way to minimizing state is to use the Integrity Protected Timestamp method described in Appendix A.

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>.

8.2. Informative References

[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-09 (work in progress), March 2018.

[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-04 (work in progress), March 2018.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[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>.

[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>.

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. Use of one of the mechanisms outlined in this section is RECOMMENDED.

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.

- Integrity Protected Timestamp. One method is to construct the Echo option value as 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. If the server loses time synchronization, e.g. due to reboot, the old key MUST be deleted and replaced by a new random secret key. A server MAY also want to encrypt its timestamps, 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

- List of Cached Random Values and Timestamps. An alternative method is to construct the Echo option value as a (pseudo-)random byte string. The server caches a list containing the random byte strings and their transmission times. Assuming 64-bit random values and 32-bit timestamps, the size of the Echo option value is 8 bytes and the amount of server state is 12n bytes, where n is the number of active Echo Option values. If the server loses time synchronization, 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

A server MAY use different methods and security levels for different uses cases (client aliveness, request freshness, state synchronization, network address reachability, etc.).

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:

- 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.]

- 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 blockwise to treat it as part of the cache-key

    * Dropped use case about OSCORE outer-blockwise (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.

Authors' Addresses

Christian Amsuess

Email: christian@amsuess.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com