ACE Working Group
Internet-Draft
Intended Status: Standards Track
Expires: April 30, 2015

G. Selander
J. Mattsson
Ericsson
L. Seitz
SICS Swedish ICT

October 27, 2014

Object Security for CoAP
draft-selander-ace-object-security-00

Abstract

We present a format for data object security in Constrained Application Protocol CoAP, i.e. protection of individual request and response messages between client and server.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Copyright (c) 2014 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 (http://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

The Constrained Application Protocol CoAP [RFC7252] was designed with a constrained RESTful environment in mind. CoAP references DTLS [RFC6347] for securing the message exchange. However, transport layer security is problematic in use cases built on store-and-forward or publish-subscribe, which require end-to-end security. DTLS offers only hop-by-hop security and requires trusted intermediaries. Moreover DTLS incurs a noticeable overhead in constrained devices due to the handshake procedure.

This memo presents an object security approach for secure messaging in constrained environments based on the protection of individual request and response messages.

In this version of the draft we focus on end-to-end integrity protection, which addresses some of the use cases e.g. where it is necessary for endpoints or proxies to verify that the message is authentic. We plan to add encryption in a later version since this is essential for other use cases.

1.1 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]. These words may also appear in this document in lowercase, absent their normative meanings.

Certain security-related terms are to be understood in the sense defined in RFC 4949 [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", "signature", and "verify".

RESTful terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252].

Terminology for constrained environments including "constrained device", "constrained-node network", "class 1", etc. are defined in [RFC7228].

Client, Resource Server, and Authorization Server are defined in [I-D.seitz-ace-problem-description]. When we just use the term server, we refer to the Resource Server.

JSON Web Signature (JWS), JOSE Header, JWS Payload, and JWS Signature are defined in [I-D.ietf-jose-json-web-signature].

NOTE: A CoAP message has header, options and payload. A JWS object has header, payload, and signature. Hence the unqualified terms "header" and "payload" have two meanings.

The JWS option is a CoAP option defined in this memo.

2. Background

The background for this work is provided by the use cases and problem description in [I-D.seitz-ace-usecases] and [I-D.seitz-ace-problem-description]. The specific part of the problem statement we address in this memo relates to sections 4.6 - 4.7 of [I-D.seitz-ace-problem-description].

The overall objective in securing access requests is that only authorized requests are granted and that the message content is protected (according to requirements of the particular use case) between client and server. As explained in the introduction, we are focusing on an efficient solution to protect requests and response end-to-end in constrained environments supporting e.g. store-and-forward use cases. To give a few examples, end-to-end integrity protection can be used to:

- o prevent manipulation and allow multiple clients to verify sensor readings stored in un-trusted intermediary nodes;

- o protect configuration data or firmware updates stored in an intermediate node, e.g. because the device was not connected at the time of the update request;

- o protect transport of authorization information ("access tokens") to sleepy devices.

The IETF has defined standardized content formats for cryptographically protected data (e.g. CMS [RFC5652], JWS [I-D.ietf-jose-json-web-signature]). Other more compact representations are in discussion in the IETF, see section 5 of [JoseWgIetf90]. One potential approach for defining data object security for constrained environments is to wrap application layer data using such a format and sending it as payload in a CoAP message. An alternative approach is to instead build data object security into the CoAP message format. The second approach is the one we propose in this memo.

As is explained in Appendix A and B this approach enables some attractive features compared to transport of protected data on top of CoAP, including:

- o Protection of certain CoAP header and option fields
- o Compliance with REST
- o Reduction of message size, by avoiding unnecessary duplication of data in payload and header/options
- o Reuse of CoAP specific mechanisms for caching and forwarding

Independently of approach, the format needs to be complemented with a description how the client and the server establish the keys, and how the keys are used for wrapping and unwrapping the secured data object. One way to address key establishment is to assume that there is a trusted third party which can support client and server, such as the Authorization Server in [I-D.draft-seitz-ace-problem-description]. The Authorization Server may, for example, authenticate the client on behalf of the server, or provide cryptographic keys or credentials to the client and/or server to secure the request/response procedure.

We emphasize that the solution sketched in this memo can be combined with DTLS [RFC6347], thus enabling end-to-end integrity protection of CoAP payload, certain CoAP headers and options, in combination with hop-by-hop protection of the entire CoAP messages during transport between end-points and intermediary devices.

3. The JWS Option

In order to integrity protect individual request and responses, as well as request-response message exchanges, we introduce a new CoAP option, the JWS option, essentially containing a digital signature or Message Authentication Code (MAC) of the CoAP message. Endpoints supporting this scheme MUST check for the presence of this option, and that the signature/MAC is valid before accepting a message as valid. The design considerations leading up to this solution are presented in Appendix A.

3.1 Option Structure

The JWS option indicates that certain CoAP header fields, options, and payload (if present) are integrity protected using JWS [I-D.ietf-jose-json-web-signature]. The JWS option SHALL contain a detached signature (JOSE Header and JWS Signature) as described in [I-D.ietf-jose-json-web-signature] Appendix F, using JWS Compact Serialization (see section 3.1 of [I-D.ietf-jose-json-web-signature]).

This option is critical, safe to forward, it is not part of a cache

key, and it is not repeatable. Table 1 illustrates the structure of this option.

No.	C	U	N	R	Name	Format	Length
TBD	x		x		JWS	opaque	125-256 B

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 1: The JWS Option

3.2 Integrity Protection and Verification

A CoAP endpoint composing a message using the JWS option SHALL process the JWS Payload and JOSE Header, defined in the following sections, according to the specification for producing the signature/MAC of a JWS object as described in Section 5.1 of the JWS specification [I-D.ietf-jose-json-web-signature].

A CoAP endpoint receiving a message containing the JWS option SHALL first recreate the JWS Payload as described in Section 3.4, and then verify the signature/MAC as defined in Section 5.2 of the JWS specification [I-D.ietf-jose-json-web-signature].

3.3 JOSE Header

Even if a signature/MAC of a received message can be verified, the message may still be old, e.g. a replay of a previous message. As is noted in section 10.10 of [I-D.ietf-jose-json-web-signature]), one way to thwart replay attacks is to include a unique message identifier and having the recipient verify that the message has not been previously received or acted upon.

As unique JWS message identifier we propose to use the combination of a unique key identifier and a sequence number. The JOSE Header of a JWS option SHALL contain either one of the "kid", "x5t", or "x5t#S256" header parameters to uniquely identify the key. In this section we define a new JOSE Header parameter "seq" (Sequence Number) enumerating the JWS objects/CoAP messages generated using the key referenced in the JOSE Header. In addition to replay protection, we want to be able to verify that a CoAP response is associated to a previously made CoAP request in order to ensure the freshness of a received response. For this purpose we require the responder to

include the sender's sequence number and key identifier in the JWS Payload.

The header field and procedures described in this section could have been replaced by similar procedures based on time-stamps, if the devices in question had reliable and synchronized clocks.

3.3.1 "seq" (Sequence Number) Header Parameter

The "seq" header parameter contains the sequence number associated to the key used to integrity protect the JWS object. The sequence number SHALL be a 32-bit number in hexadecimal representation including leading zeroes. The start sequence number SHALL be 0. For a given key, any sequence number MUST NOT be used in "seq" more than once.

The "seq" header parameter SHALL be marked as critical using the "crit" header parameter of JWS (see section 4.1.11 of [I-D.ietf-jose-json-web-signature]), meaning that if a receiver does not understand this parameter it must reject the JWS.

3.3.2 Message Sequence Numbers

In order to protect from replay and verify freshness of responses, a CoAP endpoint maintains sequence numbers.

A CoAP client supporting the JWS option SHALL store one sequence number per key it uses to protect the integrity of a message. A CoAP server supporting the JWS option SHALL store on sequence number per key it uses to verify the integrity of a message. Depending on use case, the endpoints MAY maintain a sliding receive window for sequence numbers associated to key identifiers in received messages, equivalent to the functionality described in section 4.1.2.6 of [RFC6347].

Before composing a new message with a JWS option, a CoAP client SHALL step the associated sequence number and SHALL include it in the "seq" header parameter as defined in 3.3.1. However, if the sequence number counter wraps, the client must first acquire a new key. (The latter is out of scope of this memo.)

A CoAP server supporting the JWS option SHALL verify the sequence number received in "seq" by comparing with the stored associated sequence number (or sliding window). If a CoAP server receives a valid request with a JWS option, then the response SHALL include the sequence number and key identifier of the request in the JWS Payload as defined in section 3.4.

If the CoAP client receives a response message with a JWS option, then the client SHALL generate the JWS Payload using the key identifier and the sequence number of its own associated request as defined in section 3.4

In Appendix B, we show how this can be extended to account for proxy caching functionality as well as the CoAP Observe option.

3.4 JWS Payload

The JWS Payload is type-value-length encoded and consists of:

- o the CoAP header field Code;
- o all CoAP options present which are marked as signed in Table 2 (see Appendix A); and
- o the CoAP payload (if any).
- o if the message is a response, the sequence number "seq" from the request (see section 3.3.1);
- o if the message is a response, the key identifier from the request ("kid", "x5t" or "x5t#256", see section 3.3.2)

To integrity protect the CoAP options requires the generation of a standalone representation of each option (without the option delta, see section 3.1 of [RFC7252]). The following procedure SHALL be applied to generate an option representation: Calculate the option number and represent it as a 8-bit unsigned integer. Then concatenate the 8-bit option value with a 16-bit unsigned integer in network byte order indicating the length of the Option Value, in bytes. Finally concatenate the option value (if any is present) with that bit-string.

For a request, the JWS Payload SHALL be the concatenation of the 8-bit CoAP header field Code, the CoAP option representations (as described in the previous paragraph) which are marked signed in Table 2 (see Appendix A) in the same order as given in the request, and finally a 16-bit unsigned integer in network byte order indicating the length of the CoAP payload, in bytes, and the CoAP payload of the message (if any present) as represented in the request.

For a reply, the JWS Payload SHALL be generated as above, but additionally the server SHALL append the concatenation of the 32-bit sequence number from the request, an 8-bit unsigned integer in network byte order indicating the length of the key identifier, in

bytes, and the key identifier from the request.

4. Proxy Behavior

As we target end-to-end security, we must ensure that the solution is compliant with message handling in intermediary nodes.

CoAP distinguishes between two types of proxies; forward-proxies, which are explicitly selected by clients, and reverse-proxies, which handle requests transparently to the client. Since the client is not aware of any nodes behind a reverse-proxy, it perceives the reverse-proxy as an origin server which terminates the end-to-end security.

Forward-proxies are in scope and we cover two cases here: the CoAP-CoAP forward proxy and the HTTP-CoAP cross-proxy. For CoAP-CoAP forward proxies, the JWS option SHALL be forwarded.

Using an HTTP-CoAP proxy requires that the client understands how to formulate a CoAP request. In the "Default Mapping", the Target CoAP URI is appended as-is to a base URI [I-D.ietf-core-http-mapping]. Analogously to a CoAP-CoAP forward proxy, the relevant options are copied from the HTTP URI. The JWS option SHALL be transported in the HTTP URI as a Query:

```
?JWS=...
```

where the dots "..." should be replaced by the JWS option.

Proxies not supporting the JWS option handle messages containing a JWS option according to the CoAP option processing rules, i.e. they will not process such messages themselves (since the option is marked "critical") but they will forward such messages (since the option is marked as "safe-to-forward").

5. Examples

In this section we give examples in order to illustrate and clarify the intended use of the JWS option.

5.1 GET

This example outlines a GET message exchange forwarded by a proxy. Integrity protection applies to Code, Uri-Path, Payload and other message fields.

```
    Client  Proxy  Server
       |      |      |
       |      |      |
       +----->|      |  Header: GET (Code=0.01) Token: 0x8c
       | GET  |      |  Uri-Path: "temperature"
       |      |      |  JWS: (JOSE Header: { "seq":"00000142" } ...)
       |      |      |
       |      |      |
       |      +----->|  Header: GET (Code=0.01) Token: 0x7b
       |      | GET  |  Uri-Path: "temperature"
       |      |      |  JWS: (JOSE Header: { "seq":"00000142" } ...)
       |      |      |
       |      |      |
       |      |<-----+  Header: 2.05 Content (Code=2.05) Token: 0x7b
       |      | 2.05 |  JWS: (...)
       |      |      |  Payload: "23.1 C"
       |      |      |
       |      |      |
       |<-----+      |  Header: 2.05 Content (Code=2.05) Token: 0x8c
       | 2.05 |      |  JWS: (...)
       |      |      |  Payload: "23.1 C"
       |      |      |
```

where the signature and other details are omitted. The complete JOSE header for the request is:

```
 {"alg":"HS256",
  "kid":"a1534e3c5fdc09bd",
  "crit":["seq"],
  "seq":"00000142"
 }
```

and the JWS Payload consists of:

* 000 00001 (the header field code GET)

* 0x0B (option number 11, Uri-Path)

* 0x000B (length of the option value: 11)

* "temperature" (the option value)

* (Other options are omitted for brevity.)

and for the response is:

```
 {"alg":"HS256",
```

```
  "kid":"c1a6fa909502dd82"
 }
```

The "kid" is a hint to the receiver indicating which key was used to secure the JWS, and may be used as an identifier for a secret key or a public key. It may e.g. be the hash of a public key. Even if "kid" are different in request and response, it may reference the same symmetric key.

The JWS Payload for the response consists of:

* 010 00101 (the header field code 2.05 Content)

* 0x0006 (length of the payload: 6)

* "23.1 C" (the payload value)

* "a1534e3c5fdc09bd" (the key identifier from the request)

* 0x00000142 (the sequence number from the request

5.2 POST

This example outlines a POST message exchange forwarded by a proxy. Integrity protection applies to Code, Uri-Path, Payload and other message fields.

```
 Client  Proxy  Server
    |      |      |
    |      |      |
    |      |      |
    +----->|      |  Header: POST (T=CON, Code=0.02, MID=0xf124)
    | POST |      |  Token: 0x8c
    |      |      |  Uri-Path: "lock"
    |      |      |  JWS: (JOSE Header: { "x5t":"a9095...a32a7b",
    |      |      |          "seq":"0000036f", ...} ...)
    |      |      |  Payload: "open"
    |      |      |
    |      +----->|  Header: POST (T=CON, Code=0.02, MID=0xf124)
    |      | POST |  Token: 0x8c
    |      |      |  Uri-Path: "lock"
    |      |      |  JWS: (JOSE Header: { "x5t":"a9095...a32a7b",
    |      |      |          "seq":"0000036f", ...} ...)
    |      |      |  Payload: "open"
    |      |      |
    |      |<-----+  Header: 2.04 Changed (T=ACK, Code=2.04,
    |      | 2.04 |      MID=0xf124) Token: 0x8c
    |      |      |  JWS: (JOSE Header: { "x5t":"9f2a...8520",
```

```
      |      |      |    ...} ...)
      |      |      |
      |<-----+      |  Header: 2.04 Changed (T=ACK, Code=2.04,
      | 2.04 |      |       MID=0xf124) Token: 0x8c
      |      |      |  JWS: (JOSE Header: { "x5t":"9f2a...8520",
      |      |      |    ...} ...)
      |      |      |
```

Note that in this case the client and the server are using X.509 certificates, which need to be available to both participants, so that they can look up the right public key using the thumbprint. If the proxy also has the public keys available, it can perform signature verification and discard invalid messages, in order to offload work from the client and server.

6. Security Considerations

In scenarios with proxies, gateways, or caching, DTLS only protects data hop-by-hop meaning that all intermediary nodes can modify information. The trust model where all participating nodes are considered trustworthy is problematic not only from a privacy perspective but also from a security perspective as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches makes such an architecture weak.

DTLS protects the entire CoAP message including header, options and payload, whereas this proposal only protects selected message fields. DTLS, however, also incurs a large overhead cost, due to the handshake procedure. While that cost can be amortized in scenarios with long lived connections, in cases where a device will have connections with varying clients, using secured objects instead of session security can provide a significant performance gain.

Using blockwise transfer [I-D.ietf-core-coap-block], the integrity protection as provided by the method described here only covers the individual blocks, not the entire request or response. One way to handle this would to allow the JWS option to be repeatable, and in one or several of the block transfer carry a MAC or signature that covers the entire request or response.

Since the Version header field is not integrity protected, in case of future versions of CoAP it may in theory be possible to launch a cross-version attack, e.g. something analogously to a bidding down attack. Future updates of CoAP should take this into account.

7. Privacy Considerations

End-to-end integrity protection provides certain privacy properties, e.g. protects communication with sensor and actuator from manipulation which may affect the personal sphere.

As a next step we plan to extend this scheme by add encryption for addressing other privacy concerns, such as confidentiality of personal data and prevention of pervasive monitoring.

8. IANA Considerations

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	JWS	[[this document]]

The following entries are added to the JSON Web Signature and Encryption Header Parameters registry for Header Parameter names:

- o Header Parameter Name: "seq"
- o Header Parameter Description: Message sequence number
- o Header Parameter Usage Location(s): JWS
- o Change Controller: IESG
- o Specification Document(s): Section 3.3.1 of [[this document]]]

9. References

9.1 Normative References

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and Sakimura N., "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-36 (work in progress), October 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained

Application Protocol (CoAP)", RFC 7252, June 2014.

9.2 Informative References

[I-D.seitz-ace-problem-description]
Seitz, L., and G. Selander, "Problem Description for Authorization in Constrained Environments", draft-seitz-ace-problem-description-01 (work in progress), July 2014.

[I-D.seitz-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-seitz-ace-usecases-01 (work in progress), July 2014.

[JoseWgIetf90]
Minutes of the JOSE WG meeting at IETF 90, http://www.ietf.org/proceedings/90/minutes/minutes-90-jose

[I-D.ietf-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for HTTP-CoAP Mapping Implementations", draft-ietf-core-http-mapping-05 (work in progress), October 2014.

[I-D.ietf-core-coap-block]
Bormann, C., and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-15 (work in progress), July 2014.

[I-D.ietf-jose-cookbook]
M. Miller, "Examples of Protecting Content using JavaScript Object Signing and Encryption (JOSE)", draft-ietf-jose-cookbook-05 (work in progress), October 2014.

[I-D.ietf-core-observe]
K. Hartke, "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, August 2007.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext

Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

Appendix A. Design Considerations

In this section we provide some motivation for the chosen solution. The pedagogical attempt of this section is by means of iterative modifications of the trivial solution consisting of a secure object carried in the payload.

A.1 Reducing Message Size

We noted in Section 2 that end-to-end security may be provided on application layer on top of CoAP by including, say, a JWS object [I-D.ietf-jose-json-web-signature] in the CoAP payload. The JWS represents content secured with digital signatures or Message Authentication Codes (MACs) using JavaScript Object Notation (JSON) based data structures.

However, if the content of the JWS object is independent from the CoAP message, it does not integrity protect CoAP header fields or options. To address this, one solution is to repeat certain information, contained within CoAP header fields and options, in the JWS object. However, this would not be optimal since some data would be duplicated in header/options and payload. For example, a resource identifier would be transported both as a CoAP URI-Path/URI-Query option (to comply with the CoAP message format), and in the payload (to integrity protect the intended resource which the request is targeting).

Fortunately, there is a solution to this problem known as "detached content" (Appendix F, [I-D.ietf-jose-json-web-signature]) a.k.a. "detached signature" ([I-D.ietf-jose-cookbook]). As is described in these references, the detached signature is constructed from "a JWS object in the normal fashion using a representation of the content as the payload, but then delete the payload representation from the JWS". With the outcome that "the resulting JWS object do not include the integrity protected content. Instead, the application is expected to locate it elsewhere."

Using JWS detached signature together with a specification for what message fields should be included in the digital signature or MAC, we can get integrity protection of relevant CoAP message fields without unnecessary duplication of message fields.

A.2 REST Considerations

As we saw in the previous section, a JWS detached signature in the CoAP payload would provide integrity protection and optimized message format. However, not all CoAP request and response messages support payload. E.g. GET and DELETE requests may not have defined body semantics and that could to some extent violate RESTful design. Furthermore, some CoAP response messages are not allowed to have payload or are only intended to carry resource representations.

We therefore propose to pass a JWS detached signature as a new CoAP option, as described in section 3.

NOTE: The choice of JWS is based on its relative compactness. Even compacter formats, as recently has been discussed [JoseWgIetf90], would be favorable.

A.3 Protection of CoAP Message Fields

Having motivated how a signature or MAC should be carried, we now turn to the question what information should be integrity protected.

Integrity protection should cover relevant message fields that are not supposed to change between client and server. This must also take into account that there may be intermediary devices caching and/or forwarding requests or responses.

In this section we study the message format (see Figure 1) and list the fields that need to be integrity protected as well as describe the procedure. Clearly the payload should be protected, but not all headers fields or options.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: CoAP message Format

A.3.1 CoAP Header

We now describe which fields in the CoAP header that needs to be protected.

- o Version (Ver): This field is fixed for a given implementation. However, to allow backward compatibility with future versions of CoAP, Version SHALL NOT be integrity protected.

- o Type (T) and Message ID: These fields are only relevant on CoAP messaging layer. Different Types (CON, NON, ACK, RST) or Message IDs may be used to transport the same request/response and hence SHALL NOT be integrity protected.

- o Token Length (TKL) and Token: CoAP is using the Token as a request identifier to match responses against requests. In the case of multi-hop using intermediaries, the Token may be different between the hops and is not preserved end-to-end. These fields SHALL NOT be integrity protected.

- o Code: This field is an 8-bit unsigned integer, identifying request method or response code which should not change and hence SHALL be integrity protected.

Summarizing: Only the Code header field is included in the JWS Payload of the JWS option.

A.3.2 CoAP Options

The options need to be integrity protected as follows:

- o ETag: This option defines resource local identifier of representation and hence SHALL be integrity protected.

- o If-Match, If-None-Match: These options are conditional control logic for requests which thus SHALL be integrity protected.

- o Observe: This option is elective and unsafe so may be discarded by a proxy. Hence it SHALL NOT be integrity protected.

- o Location-Path, Location-Query: These options are essentially the identifier of a new resource and hence SHALL be integrity protected.

- o Accept, Content-Format: These options indicates representation format of payload and hence SHALL be integrity protected.

- o Max-Age: The Max-Age option in the response is intended to be decreased by an intermediary device caching the response. Moreover it is elective and unsafe to forward. It SHALL NOT be integrity protected.

- o Size1: This option provides size information about the resource representation in a request and SHALL be integrity protected.

- o Proxy-Uri: This option contains the request URI, which identifies the requested resource, and hence it SHALL be integrity protected, see last item in this list.

- o Proxy-Scheme: This option contains the intended scheme to be used by a proxy, and hence it SHALL be integrity protected, see also last item in this list.

- o Uri-Host, Uri-Port, Uri-Path and Uri-Query: In a request to an origin server the request URI is decomposed into these options. In the case of requests made to an origin server, these options contain the complete information about the request URI. On the other hand in a proxy request, the request URI is specified by the client as a string in the Proxy-Uri option. The proxy which makes this a request to the origin server decomposes the Proxy-Uri into Uri-Host, Uri-Port, Uri-Path, and Uri-Query options. However, the full URI can be reconstructed at any involved endpoint.

  To allow integrity verification of the request URI, the client and forward proxies SHALL use explicit Uri-Host and Uri-Port options. The server SHALL compose the URI from options according to the method described in section 6.5 of the CoAP specification [RFC7252]. The so obtained URI is put into a Proxy-Uri option (no. 35), which is included in the integrity calculation.

Table 2 summarizes which options to include in the integrity calculation. Options marked with "x" are included. Options marked with "d" are composed into a URI as described above and included as the Proxy-Uri option for the purpose of calculating the signature. (Proxy-Uri and the options marked with "d" are mutually exclusive.)

No.	C	U	N	R	Name	Format	Length	Signed
1	x			x	If-Match	opaque	0-8	x
3	x	x	-		Uri-Host	string	1-255	d

4				x	ETag	opaque	1-8		x
5	x				If-None-Match	empty	0		x
6		x	-		Observe	uint	0-3		
7	x	x	-		Uri-Port	uint	0-2		d
8				x	Location-Path	string	0-255		x
11	x	x	-	x	Uri-Path	string	0-255		d
12					Content-Format	uint	0-2		x
14		x	-		Max-Age	uint	0-4		
15	x	x	-	x	Uri-Query	string	0-255		d
17	x				Accept	uint	0-2		x
20				x	Location-Query	string	0-255		x
35	x	x	-		Proxy-Uri	string	1-1034		x
39	x	x	-		Proxy-Scheme	string	1-255		x
60			x		Size1	uint	0-4		x
TBD	x		x		JWS	opaque	125-256		

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 2: Which options to integrity protect.

Appendix B. Replay Protection - Special Cases

In this section we show how one can use the JWS option to handle advance caching and subscribe (CoAP Observe) responses to GET requests. Please note that this is work in progress.

The general problem introduced in these settings is that there is no longer an end-to-end challenge-response protocol:

- o An intermediary forward proxies may cache a response to a corresponding GET request, and serve that response to another client's GET request.

- o A server may produce multiple responses to one GET Observe request, i.e. there is no unique matching request for each response.

This induces a number of changes:

- o In general, we can't hope to prove freshness, but can still protect from replayed responses using server sequence numbers, indicated with the "seq" header parameter.

- o However, to define an initial server sequence number we propose

to rely on an end-to-end challenge-response protocol.

- o A response message containing a challenge, is neither available nor meaningful to other clients. Since we are using both server sequence numbers and challenge-response, we need to indicate which of these freshness/replay protection parameter is used in a given response. We introduce an indicator in section B.1.

- o Since the resource identifier cannot be inferred from a default CoAP response message when there is no associated integrity protected challenge, we need to add this explicitly when we rely on server sequence numbers.

- o Note that since there may be multiple receivers of a response, this scenario makes most sense with asymmetric crypto, i.e. that the signature of the response can verified using the public key of the server.

B.1 "isi" (Integrity Scope Indication) Header Parameter

We introduce a new JOSE Header parameter indicating in requests, what freshness/replay parameter to integrity protect, and in responses, what freshness/replay protection parameter is integrity protected.

The "isi" header parameter is a 2-bit indication of what value shall be or is integrity protected in the response. The "isi" header parameter SHALL be marked as critical.

B.1.1 "isi":"01"

This indicates that the key identifier and sequence number of the request is placed in the JWS Payload of the response, and thus integrity protected. There is no server sequence number in the response. This the same procedure described in section 3.

B.1.2 "isi":"10"

This indicates that the server sequence number is in the "seq" header parameter of the response, and thus integrity protected. The key identifier and sequence number of the request is not included in the JWS payload. The response SHALL contain the request URI in the proxy-URI option.

B.1.3 "isi":"11"

This is a combination of the previous two. This indicates that the key identifier and sequence number of the request is placed in the JWS Payload, and the server sequence number is placed in the "seq"

header parameter of the response. Thus both parameters are integrity protected.

B.1.4 "isi":"00"

This value is reserved for future use.

B.2 Advance Caching

B.2.1 Acquiring server sequence numbers

```
    Client  Proxy  Server
       |      |      |
       |      |      |
       |      |      |
       +----->|      |  Header: GET (Code=0.01) Token: 0x8c
       | GET  |      |  Uri-Path: "temperature"
       |      |      |  JWS: (JOSE Header: { "kid":"b00d4272ae41433e",
       |      |      |                       "seq":"00000142",
       |      |      |                       "isi":"11",
       |      |      |                              ...} ...)
       |      |      |
       |      +----->|  Header: GET (Code=0.01) Token: 0x4b
       |      |  GET |  Uri-Path: "temperature"
       |      |      |  JWS: (JOSE Header: { "kid":"b00d4272ae41433e",
       |      |      |                       "seq":"00000142",
       |      |      |                       "isi":"11",
       |      |      |                              ...} ...)
       |      |      |
       |      |      |
       |      |<-----+  Header: 2.05 Content (Code=2.05) Token: 0x4b
       |      | 2.05 |  JWS: (JOSE Header: { "kid":"c1a6fa909502dd82",
       |      |      |                       "seq":"000000D7",
       |      |      |                       "isi":"11",
       |      |      |                              ...} ...)
       |      |      |  Payload: "23.1 C"
       |      |      |
       |      |      |
       |<-----+      |  Header: 2.05 Content (Code=2.05) Token: 0x8c
       | 2.05 |      |  JWS: (JOSE Header: { "kid":"c1a6fa909502dd82",
       |      |      |                       "seq":"000000D7",
       |      |      |                       "isi":"11",
       |      |      |                              ...} ...)
       |      |      |  Payload: "23.1 C"
       |      |      |
```

In this case, the proxy recognizes that it cannot serve a verifiably fresh cached answer to the client and therefore obtains a new one by forwarding the client's request.

The CoAP server SHALL step the associated sequence number and SHALL include it in the "seq" header parameter. However, if the sequence number counter wraps, the server must first acquire a new key. (The latter is out of scope of this memo.)

The server includes the key identifier and sequence number of the request in the JWS payload as described in section 3. The client can thus verify the freshness of the response and conclude the sequence number is fresh. Here either symmetric and asymmetric keys may be used.

B.2.2 Proxy caching

```
 Client  Proxy  Server
    |      |      |
    |      |      |
    |      +----->|  Header: GET (Code=0.01) Token: 0x4c
    |      | GET  |  Uri-Path: "temperature"
    |      |      |  JWS: (JOSE Header: { "kid":"a1534e3c5fdc09bd",
    |      |      |                       "seq":"00000070",
    |      |      |                       "isi":"10",
    |      |      |                              ...} ...)
    |      |      |
    |      |<-----+  Header: 2.05 Content (Code=2.05) Token: 0x4c
    |      | 2.05 |  JWS: (JOSE Header: { "kid":"c1a6fa909502dd82",
    |      |      |                       "seq":"000000DA",
    |      |      |                       "isi":"10",
    |      |      |                              ...} ...)
    |      |      |  Payload: "22.7 C"
    |      |      |
    |      |      |
    |      |      |
    +----->|      |  Header: GET (Code=0.01) Token: 0x8d
    | GET  |      |  Uri-Path: "temperature"
    |      |      |  JWS: (JOSE Header: { "kid":"b00d4272ae41433e",
    |      |      |                       "seq":"00000044",
    |      |      |                       "isi":"10",
    |      |      |                              ...} ...)
    |      |      |
    |<-----+      |  Header: 2.05 Content (Code=2.05) Token: 0x8d
    | 2.05 |      |  JWS: (JOSE Header: { "kid":"c1a6fa909502dd82",
    |      |      |                       "seq":"000000DA",
    |      |      |                       "isi":"10",
```

```
    |      |      |                              ...} ...)
    |      |      |  Payload: "22.7 C"
    |      |      |
```

In this case the proxy requests a response which includes the server sequence number but not the key identifier and the sequence number of the request. The response also contains the resource URI for identification of resource.

When the proxy gets a request with an "isi" header parameter that is not required to be forwarded it is matched against the cached responses, and since a corresponding response is present, it is forwarded to the client.

This setting makes most sense in the case of response "kid" identifies a public key of the server.

B.3 Observe

In certain cases, there may be more than one response associated to a request, e.g. in the case of the CoAP option Observe ([I-D.ietf-core-observe]). To securely distinguish between multiple responses and protect from replay of responses we propose the following approach:

```
Client  Server
    |      |
    |      |
    +----->|  Header: GET (Code=0.02) Token: 0x4a
    | GET  |  Uri-Path: "temperature"
    |      |  Observe: register
    |      |  JWS: (JOSE Header: { "kid":"a1534e3c5fdc09bd",
    |      |                       "seq":"0000006F",
    |      |                       "isi":"11", ...} ...)
    |      |
    |      |
    |<-----+  Header: 2.05 Content (Code=2.05) Token: 0x4a
    | 2.05 |  Observe: 12
    |      |  JWS: (JOSE Header: { "kid":"c1a6fa909502dd82",
    |      |                       "seq":"000001D6",
    |      |                       "isi":"11", ...} ...)
    |      |  Payload: "22.9 C"
    |      |
    |<-----+  Header: 2.05 Content (Code=2.05) Token: 0x4a
    | 2.05 |  Observe: 44
    |      |  JWS: (JOSE Header: { "kid":"c1a6fa909502dd82",
    |      |                       "seq":"000001D7",
    |      |                       "isi":"10", ...} ...)
```

```
       |      |  Payload: "22.8 C"
       |      |
       |<-----+  Header: 2.05 Content (Code=2.05) Token: 0x4a
       | 2.05 |  Observe: 60
       |      |  JWS: (JOSE Header: { "kid":"c1a6fa909502dd82",
       |      |                       "seq":"000001D8",
       |      |                       "isi":"10", ...} ...)
       |      |  Payload: "23.1 C"
       |      |
```

The "GET Observe: register" request SHALL contain the "isi" header parameter with value "11". The response to the "GET Observe: register" shall contain the the "isi" header parameter with value "11". This response SHALL NOT be cached. GET Observe responses without a matching request SHALL contain the the "isi" header parameter with value "10", i.e. the response SHALL contain server sequence value "seq" in JOSE Header and no request key identifier and sequence number in the JWS payload.

This procedure for replay protection of Observe also works in the presence of proxies by combining the procedures in section B.1 and B.2. This applies both to the cases of a client observing a resource through a proxy, and a proxy observing a resource to keep its cache up to date (section A.2 of [I-D.ietf-core-observe]).

Authors' Addresses

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: goran.selander@ericsson.com

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN
EMail: ludwig@sics.se

John Mattsson
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: john.mattsson@ericsson.com