

ACE
Internet-Draft
Intended status: Standards Track
Expires: March 23, 2020

P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
M. Richardson
SSW
S. Raza
RISE SICS
September 20, 2019

**EST over secure CoAP (EST-coaps)
draft-ietf-ace-coap-est-14**

Abstract

Enrollment over Secure Transport (EST) is used as a certificate provisioning protocol over HTTPS. Low-resource devices often use the lightweight Constrained Application Protocol (CoAP) for message exchanges. This document defines how to transport EST payloads over secure CoAP (EST-coaps), which allows constrained devices to use existing EST functionality for provisioning certificates.

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 March 23, 2020.

Copyright Notice

Copyright (c) 2019 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.	Change Log	3
2.	Introduction	6
3.	Terminology	7
4.	DTLS and conformance to RFC7925 profiles	7
5.	Protocol Design	9
5.1.	Discovery and URIs	10
5.2.	Mandatory/optional EST Functions	12
5.3.	Payload formats	13
5.4.	Message Bindings	14
5.5.	CoAP response codes	15
5.6.	Message fragmentation	16
5.7.	Delayed Responses	17
5.8.	Server-side Key Generation	19
6.	HTTPS-CoAPS Registrar	21
7.	Parameters	22
8.	Deployment limitations	23
9.	IANA Considerations	23
9.1.	Content-Format Registry	24
9.2.	Resource Type registry	24
10.	Security Considerations	25
10.1.	EST server considerations	25
10.2.	HTTPS-CoAPS Registrar considerations	27
11.	Contributors	28
12.	Acknowledgements	28
13.	References	28
13.1.	Normative References	28
13.2.	Informative References	30
Appendix A.	EST messages to EST-coaps	32
A.1.	cacerts	33
A.2.	enroll / reenroll	34
A.3.	serverkeygen	36
A.4.	csrattrs	38
Appendix B.	EST-coaps Block message examples	39
B.1.	cacerts	39
B.2.	enroll / reenroll	43
Appendix C.	Message content breakdown	44
C.1.	cacerts	44
C.2.	enroll / reenroll	45
C.3.	serverkeygen	47

Authors' Addresses [49](#)

1. Change Log

EDNOTE: Remove this section before publication

-14

Updates to complete Ben's AD review feedback and discussions.

-13

Updates based on AD's review and discussions

Examples redone without password

-12

Updated [section 5](#) based on Esko's comments and nits identified.

Nits and some clarifications for Esko's new review from 5/21/2019.

Nits and some clarifications for Esko's new review from 5/28/2019.

-11

Updated Server-side keygen to simplify motivation and added paragraphs in Security considerations to point out that random numbers are still needed (feedback from Hannes).

-10

Addressed WGLC comments

More consistent request format in the examples.

Explained root resource difference when there is resource discovery

Clarified when the client is supposed to do discovery

Fixed nits and minor Option length inaccuracies in the examples.

-09

WGLC comments taken into account

consensus about discovery of content-format

added additional path for content-format selection

merged DTLS sections

-08

added application/pkix-cert Content-Format TBD287.

discovery text clarified

Removed text on ct negotiation in connection to multipart-core

removed text that duplicates or contradicts [RFC7252](#) (thanks Klaus)

Stated that well-known/est is compulsory

Use of response codes clarified.

removed bugs: Max-Age and Content-Format Options in Request

Accept Option explained for est/skg and added in enroll example

Added second URI /skc for server-side key gen and a simple cert (not PKCS#7)

Persistence of DTLS connection clarified.

Minor text fixes.

-07:

redone examples from scratch with openssl

Updated authors.

Added CoAP RST as a MAY for an equivalent to an HTTP 204 message.

Added serialization example of the /skg CBOR response.

Added text regarding expired IDevIDs and persistent DTLS connection that will start using the Explicit TA Database in the new DTLS connection.

Nits and fixes

Removed CBOR envelop for binary data

Replaced TBD8 with 62.

Added [RFC8174](#) reference and text.

Clarified MTI for server-side key generation and Content-Formats.
Defined the /skg MTI (PKCS#8) and the cases where CMS encryption will be used.

Moved Fragmentation section up because it was referenced in sections above it.

-06:

clarified discovery section, by specifying that no discovery may be needed for /.well-known/est URI.

added resource type values for IANA

added list of compulsory to implement and optional functions.

Fixed issues pointed out by the idnits tool.

Updated CoAP response codes section with more mappings between EST HTTP codes and EST-coaps CoAP codes.

Minor updates to the MTI EST Functions section.

Moved Change Log section higher.

-05:

repaired again

TBD8 = 62 removed from C-F registration, to be done in CT draft.

-04:

Updated Delayed response section to reflect short and long delay options.

-03:

Removed observe and simplified long waits

Repaired Content-Format specification

-02:

Added parameter discussion in [section 8](#)

Concluded Content-Format specification using multipart-ct draft
examples updated

-01:

Editorials done.

Redefinition of proxy to Registrar in [Section 6](#). Explained better the role of https-coaps Registrar, instead of "proxy"

Provide "observe" Option examples

extended block message example.

inserted new server key generation text in [Section 5.8](#) and motivated server key generation.

Broke down details for DTLS 1.3

New Media-Type uses CBOR array for multiple Content-Format payloads

provided new Content-Format tables

new media format for IANA

-00

copied from vanderstok-ace-coap-04

2. Introduction

"Classical" Enrollment over Secure Transport (EST) [[RFC7030](#)] is used for authenticated/authorized endpoint certificate enrollment (and optionally key provisioning) through a Certificate Authority (CA) or Registration Authority (RA). EST transports messages over HTTPS.

This document defines a new transport for EST based on the Constrained Application Protocol (CoAP) since some Internet of Things (IoT) devices use CoAP instead of HTTP. Therefore, this specification utilizes DTLS [[RFC6347](#)] and CoAP [[RFC7252](#)] instead of TLS [[RFC8446](#)] and HTTP [[RFC7230](#)].

EST responses can be relatively large and for this reason this specification also uses CoAP Block-Wise Transfer [[RFC7959](#)] to offer a fragmentation mechanism of EST messages at the CoAP layer.

This document also profiles the use of EST to only support certificate-based client authentication. HTTP Basic or Digest authentication (as described in [Section 3.2.3 of \[RFC7030\]](#)) are not supported.

3. Terminology

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

Many of the concepts in this document are taken from [\[RFC7030\]](#). Consequently, much text is directly traceable to [\[RFC7030\]](#).

4. DTLS and conformance to [RFC7925](#) profiles

This section describes how EST-coaps conforms to the profiles of low-resource devices described in [\[RFC7925\]](#). EST-coaps can transport certificates and private keys. Certificates are responses to (re-)enrollment requests or requests a trusted certificate list. Private keys can be transported as responses to a server-side key generation request as described in [Section 4.4 of \[RFC7030\]](#) and discussed in [Section 5.8](#) of this document.

EST-coaps depends on a secure transport mechanism that secures the exchanged CoAP messages. DTLS is one such secure protocol. No other changes are necessary regarding the secure transport of EST messages.

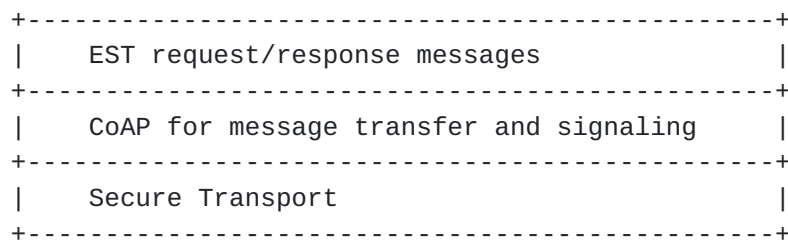


Figure 1: EST-coaps protocol layers

In accordance with sections [3.3](#) and [4.4](#) of [\[RFC7925\]](#), the mandatory cipher suite for DTLS in EST-coaps is TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 [\[RFC7251\]](#). Curve secp256r1 MUST be supported [\[RFC8422\]](#); this curve is equivalent to the NIST P-256 curve. Additionally, crypto agility is important, and the recommendations in [Section 4.4 of \[RFC7925\]](#) and any updates to it concerning Curve25519 and other curves also apply.

DTLS 1.2 implementations must use the Supported Elliptic Curves and Supported Point Formats Extensions in [RFC8422]. Uncompressed point format must also be supported. DTLS 1.3 [I-D.ietf-tls-dtls13] implementations differ from DTLS 1.2 because they do not support point format negotiation in favor of a single point format for each curve. Thus, support for DTLS 1.3 does not mandate point format extensions and negotiation. In addition, DTLS 1.3 uses the "supported_groups" extension in contrast to Supported Elliptic Curves used by DTLS 1.2

CoAP was designed to avoid IP fragmentation. DTLS is used to secure CoAP messages. However, fragmentation is still possible at the DTLS layer during the DTLS handshake when using ECC ciphersuites. If fragmentation is necessary, "DTLS provides a mechanism for fragmenting a handshake message over several records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

The authentication of the EST-coaps server by the EST-coaps client is based on certificate authentication in the DTLS handshake. The EST-coaps client MUST be configured with at least an Implicit TA database which will enable the authentication of the server the first time before updating its trust anchor (Explicit TA) [RFC7030].

The authentication of the EST-coaps client MUST be with a client certificate in the DTLS handshake. This can either be

- o a previously issued client certificate (e.g., an existing certificate issued by the EST CA); this could be a common case for simple re-enrollment of clients.
- o a previously installed certificate (e.g., manufacturer IDevID [ieee802.1ar] or a certificate issued by some other party). IDevID's are expected to have a very long life, as long as the device, but under some conditions could expire. In that case, the server MAY want to authenticate a client certificate against its trust store although the certificate is expired (Section 10).

EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in Section 3 of [RFC7030].

As described in Section 2.1 of [RFC5272] proof-of-identity refers to a value that can be used to prove that the private key corresponding to the certified public key is in the possession of and can be used by an end-entity or client. Additionally, channel-binding information can link proof-of-identity with an established connection. Connection-based proof-of-possession is OPTIONAL for EST-coaps clients and servers. When proof-of-possession is desired, a set of

actions are required regarding the use of tls-unique, described in [Section 3.5 in \[RFC7030\]](#). The tls-unique information consists of the contents of the first "Finished" message in the (D)TLS handshake between server and client [\[RFC5929\]](#). The client adds the "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request [\[RFC5967\]](#) to prove that the client is indeed in control of the private key at the time of the (D)TLS session establishment.

In the case of EST-coaps, the same operations can be performed during the DTLS handshake. For DTLS 1.2, in the event of handshake message fragmentation, the Hash of the handshake messages used in the MAC calculation of the Finished message must be computed as if each handshake message had been sent as a single fragment ([Section 4.2.6 of \[RFC6347\]](#)). The Finished message is calculated as shown in [Section 7.4.9 of \[RFC5246\]](#). Similarly, for DTLS 1.3, the Finished message must be computed as if each handshake message had been sent as a single fragment (Section 5.8 of [\[I-D.ietf-tls-dtls13\]](#)) following the algorithm described in 4.4.4 of [\[RFC8446\]](#).

In a constrained CoAP environment, endpoints can't always afford to establish a DTLS connection for every EST transaction. Authenticating and negotiating DTLS keys requires resources on low-end endpoints and consumes valuable bandwidth. To alleviate this situation, an EST-coaps DTLS connection MAY remain open for sequential EST transactions. For example, an EST csrattrs request that is followed by a simpleenroll request can use the same authenticated DTLS connection. However, when a cacerts request is included in the set of sequential EST transactions, some additional security considerations apply regarding the use of the Implicit and Explicit TA database as explained in [Section 10.1](#).

Given that after a successful enrollment, it is more likely that a new EST transaction will take place after a significant amount of time, the DTLS connections SHOULD only be kept alive for EST messages that are relatively close to each other. In some cases, like NAT rebinding, keeping the state of a connection is not possible when devices sleep for extended periods of time. In such occasions, [\[I-D.ietf-tls-dtls-connection-id\]](#) negotiates a connection ID that can eliminate the need for new handshake and its additional cost; or DTLS 1.3 session resumption provides a less costly alternative than re-doing a full DTLS handshake.

5. Protocol Design

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [\[RFC7959\]](#) to avoid IP fragmentation. The use of Blocks for

the transfer of larger EST messages is specified in [Section 5.6](#). Figure 1 shows the layered EST-coaps architecture.

The EST-coaps protocol design follows closely the EST design. The supported message types in EST-coaps are:

- o CA certificate retrieval needed to receive the complete set of CA certificates.
- o Simple enroll and re-enroll for a CA to sign client identity public key.
- o Certificate Signing Request (CSR) attribute messages that informs the client of the fields to include in a CSR.
- o Server-side key generation messages to provide a client identity private key when the client chooses so.

While [[RFC7030](#)] permits a number of the EST functions to be used without authentication, this specification requires that the client MUST be authenticated for all functions.

[5.1](#). Discovery and URIs

EST-coaps is targeted for low-resource networks with small packets. Two types of installations are possible (1) rigid ones where the address and the supported functions of the EST server(s) are known, and (2) flexible one where the EST server and its supported functions need to be discovered.

For both types of installations, saving header space is important and short EST-coaps URIs are specified in this document. These URIs are shorter than the ones in [[RFC7030](#)]. Two example EST-coaps resource path names are:

```
coaps://example.com:<port>/well-known/est/<short-est>
coaps://example.com:<port>/well-known/est/
                                ArbitraryLabel/<short-est>
```

The short-est strings are defined in Table 1. Arbitrary Labels are usually defined and used by EST CAs in order to route client requests to the appropriate certificate profile. Implementers should consider using short labels to minimize transmission overhead.

The EST-coaps server URIs, obtained through discovery of the EST-coaps resource(s) as shown below, are of the form:


```
coaps://example.com:<port>/<root-resource>/<short-est>
coaps://example.com:<port>/<root-resource>/
                                ArbitraryLabel/<short-est>
```

Figure 5 in [Section 3.2.2 of \[RFC7030\]](#) enumerates the operations and corresponding paths which are supported by EST. Table 1 provides the mapping from the EST URI path to the shorter EST-coaps URI path.

EST	EST-coaps
/cacerts	/crt
/simpleenroll	/sen
/simplereenroll	/sren
/serverkeygen	/skg (PKCS#7)
/serverkeygen	/skc (application/pkix-cert)
/csrattrs	/att

Table 1: Short EST-coaps URI path

The /skg message is the EST /serverkeygen equivalent where the client requests a certificate in PKCS#7 format and a private key. If the client prefers a single application/pkix-cert certificate instead of PKCS#7, she will make an /skc request. In both cases (i.e., /skg, /skc) a private key MUST be returned

Clients and servers MUST support the short resource EST-coaps URIs.

In the context of CoAP, the presence and location of (path to) the EST resources are discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"ace.est*"` [\[RFC6690\]](#). The example below shows the discovery over CoAPS of the presence and location of EST-coaps resources. Linefeeds are included only for readability.

```
REQ: GET /.well-known/core?rt=ace.est*
```

```
RES: 2.05 Content
```

```
</est/crt>;rt="ace.est.crt";ct="281 TBD287",
</est/sen>;rt="ace.est.sen";ct="281 TBD287",
</est/sren>;rt="ace.est.sren";ct="281 TBD287",
</est/att>;rt="ace.est.att";ct=285,
</est/skg>;rt="ace.est.skg";ct=62,
</est/skc>;rt="ace.est.skc";ct=62
```

The first three lines, describing `ace.est.crt`, `ace.est.sen`, and `ace.est.sren`, of the discovery response above MUST be returned if the

server supports resource discovery. The last three lines are only included if the corresponding EST functions are implemented (see Table 2). The Content-Formats in the response allow the client to request one that is supported by the server. These are the values that would be sent in the client request with an Accept option.

Discoverable port numbers can be returned in the response payload. An example response payload for non-default CoAPS server port 61617 follows below. Linefeeds are included only for readability.

```
REQ: GET /.well-known/core?rt=ace.est*
```

```
RES: 2.05 Content
```

```
<coaps://[2001:db8:3::123]:61617/est/crts>;rt="ace.est.crts";  
    ct="281 TBD287",  
<coaps://[2001:db8:3::123]:61617/est/sen>;rt="ace.est.sen";  
    ct="281 TBD287",  
<coaps://[2001:db8:3::123]:61617/est/sren>;rt="ace.est.sren";  
    ct="281 TBD287",  
<coaps://[2001:db8:3::123]:61617/est/att>;rt="ace.est.att";  
    ct=285,  
<coaps://[2001:db8:3::123]:61617/est/skg>;rt="ace.est.skg";  
    ct=62,  
<coaps://[2001:db8:3::123]:61617/est/skc>;rt="ace.est.skc";  
    ct=62
```

The server **MUST** support the default /.well-known/est root resource. The server **SHOULD** support resource discovery when he supports non-default URIs (like /est or /est/ArbitraryLabel) or ports. The client **SHOULD** use resource discovery when she is unaware of the available EST-coaps resources.

Throughout this document the example root resource of /est is used.

5.2. Mandatory/optional EST Functions

This specification contains a set of required-to-implement functions, optional functions, and not specified functions. The latter ones are deemed too expensive for low-resource devices in payload and calculation times.

Table 2 specifies the mandatory-to-implement or optional implementation of the EST-coaps functions. Discovery of the existence of optional functions is described in [Section 5.1](#).

EST Functions	EST-coaps implementation
/cacerts	MUST
/simpleenroll	MUST
/simplereenroll	MUST
/fullcmc	Not specified
/csrattrs	OPTIONAL
/serverkeygen	OPTIONAL

Table 2: List of EST-coaps functions

5.3. Payload formats

EST-coaps is designed for low-resource devices and hence does not need to send Base64-encoded data. Simple binary is more efficient (30% smaller payload for DER-encoded ASN.1) and well supported by CoAP. Thus, the payload for a given Media-Type follows the ASN.1 structure of the Media-Type and is transported in binary format.

The Content-Format (HTTP Media-Type equivalent) of the CoAP message determines which EST message is transported in the CoAP payload. The Media-Types specified in the HTTP Content-Type header ([Section 3.2.2 of \[RFC7030\]](#)) are specified by the Content-Format Option (12) of CoAP. The combination of URI-Path and Content-Format in EST-coaps MUST map to an allowed combination of URI and Media-Type in EST. The required Content-Formats for these requests and response messages are defined in [Section 9.1](#). The CoAP response codes are defined in [Section 5.5](#).

Content-Format TBD287 can be used in place of 281 to carry a single certificate instead of a PKCS#7 container in a /crt, /sen, /sren or /skg response. Content-Format 281 MUST be supported by EST-coaps servers. Servers MAY also support Content-Format TBD287. It is up to the client to support only Content-Format 281, TBD287 or both. The client will use a COAP Accept Option in the request to express the preferred response Content-Format. If an Accept Option is not included in the request, the client is not expressing any preference and the server SHOULD choose format 281.

Content-Format 286 is used in /sen, /sren and /skg requests and 285 in /att responses.

A representation with Content-Format identifier 62 contains a collection of representations along with their respective Content-Format. The Content-Format identifies the Media-Type application/multipart-core specified in [\[I-D.ietf-core-multipart-ct\]](#). For

example, a collection, containing two representations in response to a EST-coaps server-side key generation /skg request, could include a private key in PKCS#8 [[RFC5958](#)] with Content-Format identifier 284 (0x011C) and a single certificate in a PKCS#7 container with Content-Format identifier 281 (0x0119). Such a collection would look like [284,h'0123456789abcdef', 281,h'fedcba9876543210'] in diagnostic CBOR notation. The serialization of such CBOR content would be

```

84          # array(4)
19 011C     # unsigned(284)
48          # bytes(8)
  0123456789ABCDEF # "\x01#Eg\x89\xAB\xCD\xEF"
19 0119     # unsigned(281)
48          # bytes(8)
  FEDCBA9876543210 # "\xFE\xDC\xBA\x98vT2\x10"
```

Multipart /skg response serialization

When the client makes an /skc request the certificate returned with the private key is a single X.509 certificate (not a PKCS#7 container) with Content-Format identifier TBD287 (0x011F) instead of 281. In cases where the private key is encrypted with CMS (as explained in [Section 5.8](#)) the Content-Format identifier is 280 (0x0118) instead of 284. The content format used in the response is summarized in Table 3.

Function	Response part 1	Response part 2
/skg	284	281
/skc	280	TBD287

Table 3: response content formats for skg and skc

The key and certificate representations are ASN.1 encoded in binary format. An example is shown in [Appendix A.3](#).

5.4. Message Bindings

The general EST-coaps message characteristics are:

- o EST-coaps servers sometimes need to provide delayed responses which are preceded by an immediately returned empty ACK or an ACK containing response code 5.03 as explained in [Section 5.7](#). Thus, it is RECOMMENDED for implementers to send EST-coaps requests in confirmable CON CoAP messages.

- o The CoAP Options used are Uri-Host, Uri-Path, Uri-Port, Content-Format, Block1, Block2, and Accept. These CoAP Options are used to communicate the HTTP fields specified in the EST REST messages. The Uri-host and Uri-Port Options can be omitted from the COAP message sent on the wire. When omitted, they are logically assumed to be the transport protocol destination address and port respectively. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly. Other COAP Options should be handled in accordance with [\[RFC7252\]](#).
- o EST URLs are HTTPS based (https://), in CoAP these are assumed to be translated to CoAPS (coaps://)

Table 1 provides the mapping from the EST URI path to the EST-coaps URI path. [Appendix A](#) includes some practical examples of EST messages translated to CoAP.

5.5. CoAP response codes

[Section 5.9 of \[RFC7252\]](#) and [Section 7 of \[RFC8075\]](#) specify the mapping of HTTP response codes to CoAP response codes. The success code in response to an EST-coaps GET request (/crt, /att), is 2.05. Similarly, 2.04 is used in successful response to EST-coaps POST requests (/sen, /sren, /skg, /skc).

EST makes use of HTTP 204 or 404 responses when a resource is not available for the client. In EST-coaps 2.04 is used in response to a POST (/sen, /sren, /skg, /skc). 4.04 is used when the resource is not available for the client.

HTTP response code 202 with a Retry-After header in [\[RFC7030\]](#) has no equivalent in CoAP. HTTP 202 with Retry-After is used in EST for delayed server responses. [Section 5.7](#) specifies how EST-coaps handles delayed messages with 5.03 responses with a Max-Age Option.

Additionally, EST's HTTP 400, 401, 403, 404 and 503 status codes have their equivalent CoAP 4.00, 4.01, 4.03, 4.04 and 5.03 response codes in EST-coaps. Table 4 summarizes the EST-coaps response codes.

operation	EST-coaps response code	Description
/crt, /att	2.05	Success. Certs included in the response payload.
	4.xx / 5.xx	Failure.
/sen, /skg, /sren, /skc	2.04	Success. Cert included in the response payload.
	5.03	Retry in Max-Age Option time.
	4.xx / 5.xx	Failure.

Table 4: EST-coaps response codes

5.6. Message fragmentation

DTLS defines fragmentation only for the handshake and not for secure data exchange (DTLS records). [RFC6347] states that to avoid using IP fragmentation, which involves error-prone datagram reconstitution, invokers of the DTLS record layer should size DTLS records so that they fit within any Path MTU estimates obtained from the record layer. In addition, invokers residing on a 6LoWPAN over IEEE 802.15.4 [ieee802.15.4] network are recommended to size CoAP messages such that each DTLS record will fit within one or two IEEE 802.15.4 frames.

That is not always possible in EST-coaps. Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and Object Identifier (OID) fields used. For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, Subject Alternative Names (SAN) and cert fields. For 384-bit curves, ECDSA certificates increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacerts response from the server can include multiple certificates that amount to large payloads. Section 4.6 of CoAP [RFC7252] describes the possible payload sizes: "if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size". Section 4.6 of [RFC7252] also suggests that IPv4 implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes. Even with ECC, EST-coaps messages can still exceed MTU sizes on the Internet or 6LoWPAN [RFC4919] (Section 2 of [RFC7959]). EST-coaps needs to be able to fragment messages into multiple DTLS datagrams.

To perform fragmentation in CoAP, [RFC7959] specifies the Block1 Option for fragmentation of the request payload and the Block2 Option

for fragmentation of the return payload of a CoAP flow. As explained in [Section 1 of \[RFC7959\]](#), block-wise transfers should be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks. Both EST-coaps clients and servers MUST support Block2. EST-coaps servers MUST also support Block1. The EST-coaps client MUST support Block1 only if it sends EST-coaps requests with an IP packet size that exceeds the Path MTU.

[RFC7959] also defines Size1 and Size2 Options to provide size information about the resource representation in a request and response. EST-client and server MAY support Size1 and Size2 Options.

Examples of fragmented EST-coaps messages are shown in [Appendix B](#).

5.7. Delayed Responses

Server responses can sometimes be delayed. According to [Section 5.2.2 of \[RFC7252\]](#), a slow server can acknowledge the request and respond later with the requested resource representation. In particular, a slow server can respond to an EST-coaps enrollment request with an empty ACK with code 0.00, before sending the certificate to the client after a short delay. If the certificate response is large, the server will need more than one Block2 block to transfer it.

This situation is shown in Figure 2. The client sends an enrollment request that uses $N1+1$ Block1 blocks. The server uses an empty 0.00 ACK to announce the delayed response which is provided later with 2.04 messages containing $N2+1$ Block2 Options. The first 2.04 is a confirmable message that is acknowledged by the client. Onwards, having received the first 256 bytes in the first Block2 block, the client asks for a block reduction to 128 bytes in a confirmable enrollment request and acknowledges the Block2 blocks sent up to that point.

The notation of Figure 2 is explained in [Appendix B.1](#).


```

POST [2001:db8::2:1]:61616/est/sen (CON)(1:0/1/256) {CSR (frag# 1)} -->
  <-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR (frag# 2)} -->
  <-- (ACK) (1:1/1/256) (2.31 Continue)
      .
      .
      .
POST [2001:db8::2:1]:61616/est/sen(CON)(1:N1/0/256){CSR (frag# N1+1)}-->
  <-- (0.00 empty ACK)
      |
  ... Short delay before the certificate is ready ...
      |
  <-- (CON) (1:N1/0/256)(2:0/1/256)(2.04 Changed) {Cert resp (frag# 1)}
                                (ACK)                                -->
POST [2001:db8::2:1]:61616/est/sen (CON)(2:1/0/128)                                -->
  <-- (ACK) (2:1/1/128) (2.04 Changed) {Cert resp (frag# 2)}
      .
      .
      .
POST [2001:db8::2:1]:61616/est/sen (CON)(2:N2/0/128)                                -->
  <-- (ACK) (2:N2/0/128) (2.04 Changed) {Cert resp (frag# N2+1)}

```

Figure 2: EST-COAP enrollment with short wait

If the server is very slow (i.e., minutes) in providing the response (i.e., when a manual intervention is needed), he SHOULD respond with an ACK containing response code 5.03 (Service unavailable) and a Max-Age Option to indicate the time the client SHOULD wait to request the content later. After a delay of Max-Age, the client SHOULD resend the identical CSR to the server. As long as the server responds with response code 5.03 (Service Unavailable) with a Max-Age Option, the client SHOULD keep resending the enrollment request until the server responds with the certificate or the client abandons the request for other reasons.

To demonstrate this scenario, Figure 3 shows a client sending an enrollment request that uses N1+1 Block1 blocks to send the CSR to the server. The server needs N2+1 Block2 blocks to respond, but also needs to take a long delay (minutes) to provide the response. Consequently, the server uses a 5.03 ACK response with a Max-Age Option. The client waits for a period of Max-Age as many times as she receives the same 5.03 response and retransmits the enrollment request until she receives a certificate in a fragmented 2.04 response. Note that the client asks again for a decrease in the block size when acknowledging the first Block2.


```

POST [2001:db8::2:1]:61616/est/sen (CON)(1:0/1/256) {CSR (frag# 1)} -->
<-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR (frag# 2)} -->
<-- (ACK) (1:1/1/256) (2.31 Continue)
.
.
.
POST [2001:db8::2:1]:61616/est/sen(CON)(1:N1/0/256){CSR (frag# N1+1)}-->
<-- (ACK) (1:N1/0/256) (5.03 Service Unavailable) (Max-Age)
|
|
... Client tries again after Max-Age with identical payload ...
|
|
POST [2001:db8::2:1]:61616/est/sen(CON)(1:0/1/256){CSR (frag# 1)}-->
<-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR (frag# 2)} -->
<-- (ACK) (1:1/1/256) (2.31 Continue)
.
.
.
POST [2001:db8::2:1]:61616/est/sen(CON)(1:N1/0/256){CSR (frag# N1+1)}-->
|
... Immediate response when certificate is ready ...
|
<-- (ACK) (1:N1/0/256) (2:0/1/256) (2.04 Changed){Cert resp (frag# 1)}
POST [2001:db8::2:1]:61616/est/sen (CON)(2:1/0/128) -->
<-- (ACK) (2:1/1/128) (2.04 Changed) {Cert resp (frag# 2)}
.
.
.
POST [2001:db8::2:1]:61616/est/sen (CON)(2:N2/0/128) -->
<-- (ACK) (2:N2/0/128) (2.04 Changed) {Cert resp (frag# N2+1)}

```

Figure 3: EST-COAP enrollment with long wait

5.8. Server-side Key Generation

In scenarios where it is desirable that the server generates the private key, server-side key generation is available. Such scenarios could be when it is considered more secure to generate at the server the long-lived random private key that identifies the client, or when the resources spent to generate a random private key at the client are considered scarce, or when the security policy requires that the certificate public and corresponding private keys are centrally generated and controlled. Of course, that does not eliminate the need for proper random numbers in various protocols like (D)TLS ([Section 10.1](#)).

When requesting server-side key generation, the client asks for the server or proxy to generate the private key and the certificate which are transferred back to the client in the server-side key generation response. In all respects, the server treats the CSR as it would treat any enroll or re-enroll CSR; the only distinction here is that the server **MUST** ignore the public key values and signature in the CSR. These are included in the request only to allow re-use of existing codebases for generating and parsing such requests.

The client /skg request is for a certificate in a PKCS#7 container and private key in two application/multipart-core elements. Respectively, an /skc request is for a single application/pkix-cert certificate and a private key. The private key Content-Format requested by the client is indicated in the PKCS#10 CSR request. If the request contains SMIMECapabilities and DecryptKeyIdentifier or AsymmetricDecryptKeyIdentifier the client is expecting Content-Format 280 for the private key. Then the private key is encrypted symmetrically or asymmetrically as per [\[RFC7030\]](#). The symmetric key or the asymmetric keypair establishment method is out of scope of the specification. A /skg or /skc request with a CSR without SMIMECapabilities expects an application/multipart-core with an unencrypted PKCS#8 private key with Content-Format 284.

The EST-coaps server-side key generation response is returned with Content-Format application/multipart-core [\[I-D.ietf-core-multipart-ct\]](#) containing a CBOR array with four items ([Section 5.3](#)). The two representations (each consisting of two CBOR array items) are preceded by its Content-Format ID. Dependent on the request, the private key can be in unprotected PKCS#8 [\[RFC5958\]](#) format (Content-Format 284) or protected inside of CMS SignedData (Content-Format 280). The SignedData, placed in the outermost container, is signed by the party that generated the private key, which may be the EST server or the EST CA. SignedData placed within the Enveloped Data does not need additional signing as explained in [Section 4.4.2 of \[RFC7030\]](#). In summary, the symmetrically encrypted key is included in the encryptedKey attribute in a KEKRecipientInfo structure. In the case where the asymmetric encryption key is suitable for transport key operations the generated private key is encrypted with a symmetric key which is encrypted by the client-defined (in the CSR) asymmetric public key and is carried in an encryptedKey attribute in a KeyTransRecipientInfo structure. Finally, if the asymmetric encryption key is suitable for key agreement, the generated private key is encrypted with a symmetric key which is encrypted by the client defined (in the CSR) asymmetric public key and is carried in an recipientEncryptedKeys attribute in a KeyAgreeRecipientInfo.

[RFC7030] recommends the use of additional encryption of the returned private key. For the context of this specification, clients and servers that choose to support server-side key generation MUST support unprotected (PKCS#8) private keys (Content-Format 284). Symmetric or asymmetric encryption of the private key (CMS EnvelopedData, Content-Format 280) SHOULD be supported for deployments where end-to-end encryption is needed between the client and a server. Such cases could include architectures where an entity between the client and the CA terminates the DTLS connection (Registrar in Figure 4). This document does not strongly recommend CMS encryption on top of the DTLS channel like [RFC7030] unless mandated by the use-case.

6. HTTPS-CoAPS Registrar

In real-world deployments, the EST server will not always reside within the CoAP boundary. The EST server can exist outside the constrained network in which case it will support TLS/HTTP instead of CoAPS. In such environments EST-coaps is used by the client within the CoAP boundary and TLS is used to transport the EST messages outside the CoAP boundary. A Registrar at the edge is required to operate between the CoAP environment and the external HTTP network as shown in Figure 4.

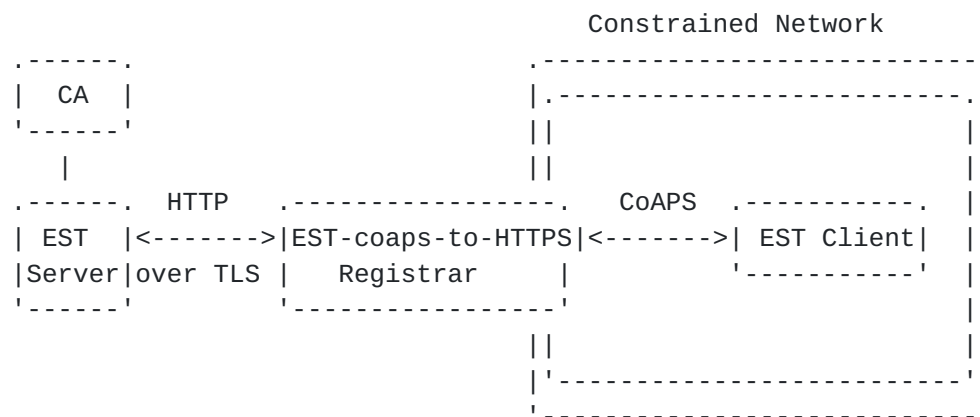


Figure 4: EST-coaps-to-HTTPS Registrar at the CoAP boundary.

The EST-coaps-to-HTTPS Registrar MUST terminate EST-coaps downstream and initiate EST connections over TLS upstream. The Registrar MUST authenticate and optionally authorize the client requests while it MUST be authenticated by the EST server or CA. The trust relationship between the Registrar and the EST server SHOULD be pre-established for the Registrar to proxy these connections on behalf of various clients.

When enforcing Proof-of-Possession (PoP) linking, the DTLS `tls-unique` value of the (D)TLS session is used to prove that the private key corresponding to the public key is in the possession of the client and was used to establish the connection as explained in [Section 4](#). The PoP linking information is lost between the EST-coaps client and the EST server when a Registrar is present. The EST server becomes aware of the presence of a Registrar from its TLS client certificate that includes `id-kp-cmcRA` [[RFC6402](#)] extended key usage extension (EKU). As explained in [Section 3.7 of \[RFC7030\]](#), the "EST server SHOULD apply an authorization policy consistent with a Registrar client. For example, it could be configured to accept PoP linking information that does not match the current TLS session because the authenticated EST client Registrar has verified this information when acting as an EST server".

For some use-cases, clients that leverage server-side key generation might prefer for the enrolled keys to be generated by the Registrar if the CA does not support server-side key generation. Such a Registrar is responsible for generating a new CSR signed by a new key which will be returned to the client along with the certificate from the CA. In these cases, the Registrar MUST use random number generation with proper entropy.

Table 1 contains the URI mappings between EST-coaps and EST that the Registrar MUST adhere to. [Section 5.5](#) of this specification and [Section 7 of \[RFC8075\]](#) define the mappings between EST-coaps and HTTP response codes, that determine how the Registrar MUST translate CoAP response codes from/to HTTP status codes. The mapping from CoAP Content-Format to HTTP Media-Type is defined in [Section 9.1](#). Additionally, a conversion from CBOR major type 2 to Base64 encoding MUST take place at the Registrar. If CMS end-to-end encryption is employed for the private key, the encrypted CMS EnvelopedData blob MUST be converted at the Registrar to binary CBOR type 2 downstream to the client.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP Registrar MUST reassemble the BLOCKs before translating the binary content to Base64, and consecutively relay the message upstream.

The EST-coaps-to-HTTP Registrar MUST support resource discovery according to the rules in [Section 5.1](#).

7. Parameters

This section addresses transmission parameters described in sections 4.7 and 4.8 of [[RFC7252](#)]. EST does not impose any unique values on the CoAP parameters in [[RFC7252](#)], but the setting of the CoAP

parameter values may have consequence for the setting of the EST parameter values.

It is recommended, based on experiments, to follow the default CoAP configuration parameters ([RFC7252]). However, depending on the implementation scenario, retransmissions and timeouts can also occur on other networking layers, governed by other configuration parameters. When a change in a server parameter has taken place, the parameter values in the communicating endpoints MUST be adjusted as necessary.

Some further comments about some specific parameters, mainly from Table 2 in [RFC7252]:

- o NSTART: A parameter that controls the number of simultaneous outstanding interactions that a client maintains to a given server. An EST-coaps client is expected to control at most one interaction with a given server, which is the default NSTART value defined in [RFC7252].
- o DEFAULT_LEISURE: This setting is only relevant in multicast scenarios, outside the scope of EST-coaps.
- o PROBING_RATE: A parameter which specifies the rate of re-sending non-confirmable messages. In the rare situations that non-confirmable messages are used, the default PROBING_RATE value defined in [RFC7252] applies.

Finally, the Table 3 parameters in [RFC7252] are mainly derived from Table 2. Directly changing parameters on one table would affect parameters on the other.

8. Deployment limitations

Although EST-coaps paves the way for the utilization of EST by constrained devices in constrained networks, some classes of devices [RFC7228] will not have enough resources to handle the payloads that come with EST-coaps. The specification of EST-coaps is intended to ensure that EST works for networks of constrained devices that choose to limit their communications stack to DTLS/CoAP. It is up to the network designer to decide which devices execute the EST protocol and which do not.

9. IANA Considerations

9.1. Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry [[COREparams](#)] are specified in Table 5. These have been registered provisionally in the IETF Review or IESG Approval range (256-9999).

HTTP Media-Type	ID	Reference
application/pkcs7-mime; smime-type=server-generated- key	280	[RFC7030] [I-D.ietf-lamps- rfc5751-bis] [ThisRFC]
application/pkcs7-mime; smime-type=certs-only	281	[I-D.ietf-lamps-rfc5751-bi s] [ThisRFC]
application/pkcs8	284	[RFC5958] [I-D.ietf-lamps- rfc5751-bis] [ThisRFC]
application/csrattrs	285	[RFC7030]
application/pkcs10	286	[RFC5967] [I-D.ietf-lamps- rfc5751-bis] [ThisRFC]
application/pkix-cert	TBD28 7	[RFC2585] [ThisRFC]

Table 5: New CoAP Content-Formats

It is suggested that 287 is allocated to TBD287.

9.2. Resource Type registry

This memo registers new Resource Type (rt=) Link Target Attributes in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

- o rt="ace.est.certs". This resource depicts the support of EST get cacerts.
- o rt="ace.est.sen". This resource depicts the support of EST simple enroll.
- o rt="ace.est.sren". This resource depicts the support of EST simple reenroll.
- o rt="ace.est.att". This resource depicts the support of EST get CSR attributes.

- o `rt="ace.est.skg"`. This resource depicts the support of EST server-side key generation with the returned certificate in a PKCS#7 container.
- o `rt="ace.est.skc"`. This resource depicts the support of EST server-side key generation with the returned certificate in application/pkix-cert format.

10. Security Considerations

10.1. EST server considerations

The security considerations of [Section 6 of \[RFC7030\]](#) are only partially valid for the purposes of this document. As HTTP Basic Authentication is not supported, the considerations expressed for using passwords do not apply. The other portions of the security considerations of [\[RFC7030\]](#) continue to apply.

Modern security protocols require random numbers to be available during the protocol run, for example for nonces and ephemeral (EC) Diffie-Hellman key generation. This capability to generate random numbers is also needed when the constrained device generates the private key (that corresponds to the public key enrolled in the CSR). When server-side key generation is used, the constrained device depends on the server to generate the private key randomly, but it still needs locally generated random numbers for use in security protocols, as explained in [Section 12 of \[RFC7925\]](#). Additionally, the transport of keys generated at the server is inherently risky. For those deploying server-side key generation, analysis SHOULD be done to establish whether server-side key generation increases or decreases the probability of digital identity theft.

It is important to note that sources contributing to the randomness pool used to generate random numbers on laptops or desktop PCs are not available on many constrained devices, such as mouse movement, timing of keystrokes, or air turbulence on the movement of hard drive heads, as pointed out in [\[PsQs\]](#). Other sources have to be used or dedicated hardware has to be added. Selecting hardware for an IoT device that is capable of producing high-quality random numbers is therefore important [\[RSAfact\]](#).

It is also RECOMMENDED that the Implicit Trust Anchor database used for EST server authentication is carefully managed to reduce the chance of a third-party CA with poor certification practices jeopardizing authentication. Disabling the Implicit Trust Anchor database after successfully receiving the Distribution of CA certificates response ([Section 4.1.3 of \[RFC7030\]](#)) limits any risk to the first DTLS exchange. Alternatively, in a case where a /sen

request immediately follows a /crt, a client MAY choose to keep the connection authenticated by the Implicit TA open for efficiency reasons ([Section 4](#)). A client that interleaves EST-coaps /crt request with other requests in the same DTLS connection SHOULD revalidate the server certificate chain against the updated Explicit TA from the /crt response before proceeding with the subsequent requests. If the server certificate chain does not authenticate against the database, the client SHOULD close the connection without completing the rest of the requests. The updated Explicit TA MUST continue to be used in new DTLS connections.

In cases where the IDevID used to authenticate the client is expired the server MAY still authenticate the client because IDevIDs are expected to live as long as the device itself ([Section 4](#)). In such occasions, checking the certificate revocation status or authorizing the client using another method is important for the server to ensure that the client is to be trusted.

In accordance with [\[RFC7030\]](#), TLS cipher suites that include "_EXPORT_" and "_DES_" in their names MUST NOT be used. More information about recommendations of TLS and DTLS are included in [\[RFC7525\]](#).

As described in CMC, [Section 6.7 of \[RFC5272\]](#), "For keys that can be used as signature keys, signing the certification request with the private key serves as a PoP on that key pair". The inclusion of tls-unique in the certificate request links the proof-of-possession to the TLS proof-of-identity. This implies but does not prove that only the authenticated client currently has access to the private key.

What's more, CMC PoP linking uses tls-unique as it is defined in [\[RFC5929\]](#). The 3SHAKE attack [[tripleshake](#)] poses a risk by allowing a man-in-the-middle to leverage session resumption and renegotiation to inject himself between a client and server even when channel binding is in use. In the context of this specification, an attacker could invalidate the purpose of the PoP linking ChallengePassword in the client request by resuming an EST-coaps connection. Even though the practical risk of such an attack to EST-coaps is not devastating, we would rather use a more secure channel binding mechanism. Such a mechanism could include an updated tls-unique value generation like the tls-unique-prf defined in [[I-D.josefsson-sasl-tls-cb](#)] by using a TLS exporter [[RFC5705](#)] in TLS 1.2 or TLS 1.3's updated exporter ([Section 7.5 of \[RFC8446\]](#)) value in place of the tls-unique value in the CSR. Such mechanism has not been standardized yet. Adopting a channel binding value generated from an exporter would break backwards compatibility for an RA that proxies through to a classic EST server. Thus, in this specification we still depend on the tls-

unique mechanism defined in [[RFC5929](#)], especially since a 3SHAKE attack does not expose messages exchanged with EST-coaps.

Regarding the Certificate Signing Request (CSR), an EST-coaps server is expected to be able to recover from improper CSR requests.

Interpreters of ASN.1 structures should be aware of the use of invalid ASN.1 length fields and should take appropriate measures to guard against buffer overflows, stack overruns in particular, and malicious content in general.

10.2. HTTPS-CoAPS Registrar considerations

The Registrar proposed in [Section 6](#) must be deployed with care, and only when direct client-server connections are not possible. When PoP linking is used the Registrar terminating the DTLS connection establishes a new TLS connection with the upstream CA. Thus, it is impossible for PoP linking to be enforced end-to-end for the EST transaction. The EST server could be configured to accept PoP linking information that does not match the current TLS session because the authenticated EST Registrar is assumed to have verified PoP linking downstream to the client.

The introduction of an EST-coaps-to-HTTP Registrar assumes the client can authenticate the Registrar using its implicit or explicit TA database. It also assumes the Registrar has a trust relationship with the upstream EST server in order to act on behalf of the clients. When a client uses the Implicit TA database for certificate validation, she SHOULD confirm if the server is acting as an RA by the presence of the id-kp-cmcRA EKU [[RFC6402](#)] in the server certificate.

In a server-side key generation case, if no end-to-end encryption is used, the Registrar may be able see the private key as it acts as a man-in-the-middle. Thus, the client puts its trust on the Registrar not exposing the private key.

Clients that leverage server-side key generation without end-to-end encryption of the private key ([Section 5.8](#)) have no knowledge if the Registrar will be generating the private key and enrolling the certificates with the CA or if the CA will be responsible for generating the key. In such cases, the existence of a Registrar requires the client to put its trust on the registrar when it is generating the private key.

11. Contributors

Martin Furuheid contributed to the EST-coaps specification by providing feedback based on the Nexus EST over CoAPS server implementation that started in 2015. Sandeep Kumar kick-started this specification and was instrumental in drawing attention to the importance of the subject.

12. Acknowledgements

The authors are very grateful to Klaus Hartke for his detailed explanations on the use of Block with DTLS and his support for the Content-Format specification. The authors would like to thank Esko Dijk and Michael Verschoor for the valuable discussions that helped in shaping the solution. They would also like to thank Peter Panburana for his feedback on technical details of the solution. Constructive comments were received from Benjamin Kaduk, Eliot Lear, Jim Schaad, Hannes Tschofenig, Julien Vermillard, John Manuel, Oliver Pfaff, Pete Beal and Carsten Bormann.

Interop tests were done by Oliver Pfaff, Thomas Werner, Oskar Camezind, Bjorn Elmers and Joel Hoglund.

Robert Moskowitz provided code to create the examples.

13. References

13.1. Normative References

- [I-D.ietf-core-multipart-ct]
Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for CoAP", [draft-ietf-core-multipart-ct-04](#) (work in progress), August 2019.
- [I-D.ietf-lamps-rfc5751-bis]
Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", [draft-ietf-lamps-rfc5751-bis-12](#) (work in progress), September 2018.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", [draft-ietf-tls-dtls13-32](#) (work in progress), July 2019.

- [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>>.
- [RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", [RFC 2585](#), DOI 10.17487/RFC2585, May 1999, <<https://www.rfc-editor.org/info/rfc2585>>.
- [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>>.
- [RFC5958] Turner, S., "Asymmetric Key Packages", [RFC 5958](#), DOI 10.17487/RFC5958, August 2010, <<https://www.rfc-editor.org/info/rfc5958>>.
- [RFC5967] Turner, S., "The application/pkcs10 Media Type", [RFC 5967](#), DOI 10.17487/RFC5967, August 2010, <<https://www.rfc-editor.org/info/rfc5967>>.
- [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>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", [RFC 6690](#), DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", [RFC 7030](#), DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.
- [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>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", [RFC 7925](#), DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.

- [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>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", [RFC 8075](#), DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", [RFC 8422](#), DOI 10.17487/RFC8422, August 2018, <<https://www.rfc-editor.org/info/rfc8422>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [RFC 8446](#), DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

13.2. Informative References

- [COREparams]
"Constrained RESTful Environments (CoRE) Parameters", <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml>>.
- [I-D.ietf-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", [draft-ietf-tls-dtls-connection-id-06](#) (work in progress), July 2019.
- [I-D.josefsson-sasl-tls-cb]
Josefsson, S., "Channel Bindings for TLS based on the PRF", [draft-josefsson-sasl-tls-cb-03](#) (work in progress), March 2015.
- [I-D.moskowitz-ecdsa-pki]
Moskowitz, R., Birkholz, H., Xia, L., and M. Richardson, "Guide for building an ECC pki", [draft-moskowitz-ecdsa-pki-07](#) (work in progress), August 2019.

- [ieee802.15.4]
"IEEE Standard 802.15.4-2006", 2006.
- [ieee802.1ar]
"IEEE 802.1AR Secure Device Identifier", December 2009.
- [PsQs] "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices", USENIX Security Symposium 2012 ISBN 978-931971-95-9, August 2012.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", [RFC 4919](#), DOI 10.17487/RFC4919, August 2007, <<https://www.rfc-editor.org/info/rfc4919>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", [RFC 5272](#), DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", [RFC 5705](#), DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", [RFC 5929](#), DOI 10.17487/RFC5929, July 2010, <<https://www.rfc-editor.org/info/rfc5929>>.
- [RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", [RFC 6402](#), DOI 10.17487/RFC6402, November 2011, <<https://www.rfc-editor.org/info/rfc6402>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", [RFC 7228](#), DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", [RFC 7251](#), DOI 10.17487/RFC7251, June 2014, <<https://www.rfc-editor.org/info/rfc7251>>.

- [RFC7299] Housley, R., "Object Identifier Registry for the PKIX Working Group", [RFC 7299](#), DOI 10.17487/RFC7299, July 2014, <<https://www.rfc-editor.org/info/rfc7299>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", [BCP 195](#), [RFC 7525](#), DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RSAfact] "Factoring RSA keys from certified smart cards: Coppersmith in the wild", Advances in Cryptology - ASIACRYPT 2013, August 2013.
- [tripleshake]
"Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS", IEEE Security and Privacy ISBN 978-1-4799-4686-0, May 2014.

[Appendix A](#). EST messages to EST-coaps

This section shows similar examples to the ones presented in [Appendix A of \[RFC7030\]](#). The payloads in the examples are the hex encoded binary, generated with 'xxd -p', of the PKI certificates created following [\[I-D.moskowitz-ecdsa-pki\]](#). Hex is used for visualization purposes because a binary representation cannot be rendered well in text. The hexadecimal representations would not be transported in hex, but in binary. The payloads are shown unencrypted. In practice the message content would be transferred over an encrypted DTLS channel.

The certificate responses included in the examples contain Content-Format 281 (application/pkcs7). If the client had requested Content-Format TBD287 (application/pkix-cert) by querying /est/skc, the server would respond with a single DER binary certificate in the multipart-core container.

These examples assume a short resource path of "/est". Even though omitted from the examples for brevity, before making the EST-coaps requests, a client would learn about the server supported EST-coaps resources with a GET request for /.well-known/core?rt=ace.est* as explained in [Section 5.1](#).

The corresponding CoAP headers are only shown in [Appendix A.1](#). Creating CoAP headers is assumed to be generally understood.

The message content breakdown is presented in [Appendix C](#).

[A.1.](#) cacerts

In EST-coaps, a cacerts message can be:

```
GET example.com:9085/est/crts
(Accept: 281)
```

The corresponding CoAP header fields are shown below. The use of block and DTLS are worked out in [Appendix B](#).

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.01 is GET)
Token = 0x9a (client generated)
Options
Option (Uri-Host)
  Option Delta = 0x3 (option# 3)
  Option Length = 0xB
  Option Value = "example.com"
Option (Uri-Port)
  Option Delta = 0x4 (option# 3+4=7)
  Option Length = 0x2
  Option Value = 9085
Option (Uri-Path)
  Option Delta = 0x4 (option# 7+4=11)
  Option Length = 0x3
  Option Value = "est"
Option (Uri-Path)
  Option Delta = 0x0 (option# 11+0=11)
  Option Length = 0x4
  Option Value = "crts"
Option (Accept)
  Option Delta = 0x6 (option# 11+6=17)
  Option Length = 0x2
  Option Value = 281
Payload = [Empty]
```

The Uri-Host and Uri-Port Options can be omitted if they coincide with the transport protocol destination address and port respectively. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly.

A 2.05 Content response with a cert in EST-coaps will then be

```
2.05 Content (Content-Format: 281)
  {payload with certificate in binary format}
```


with CoAP fields

```
Ver = 1
T = 2 (ACK)
Code = 0x45 (2.05 Content)
Token = 0x9a (copied from request by server)
Options
  Option (Content-Format)
    Option Delta = 0xC (option# 12)
    Option Length = 0x2
    Option Value = 281
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

Payload =

```
3082027a06092a864886f70d010702a082026b308202670201013100300b
06092a864886f70d010701a082024d30820249308201efa0030201020208
0b8bb0fe604f6a1e300a06082a8648ce3d0403023067310b300906035504
0613025553310b300906035504080c024341310b300906035504070c024c
4131143012060355040a0c0b4578616d706c6520496e6331163014060355
040b0c0d63657274696669636174696f6e3110300e06035504030c07526f
6f74204341301e170d3139303133313131323730335a170d333930313236
3131323730335a3067310b3009060355040613025553310b300906035504
080c024341310b300906035504070c024c4131143012060355040a0c0b45
78616d706c6520496e6331163014060355040b0c0d636572746966696361
74696f6e3110300e06035504030c07526f6f742043413059301306072a86
48ce3d020106082a8648ce3d030107034200040c1b1e82ba8cc72680973f
97edb8a0c72ab0d405f05d4fe29b997a14ccce89008313d09666b6ce375c
595fcc8e37f8e4354497011be90e56794bd91ad951ab45a3818430818130
1d0603551d0e041604141df1208944d77b5f1d9dcb51ee244a523f3ef5de
301f0603551d230418301680141df1208944d77b5f1d9dcb51ee244a523f
3ef5de300f0603551d130101ff040530030101ff300e0603551d0f0101ff
040403020106301e0603551d110417301581136365727469667940657861
6d706c652e636f6d300a06082a8648ce3d040302034800304502202b891d
d411d07a6d6f621947635ba4c43165296b3f633726f02e51ecf464bd4002
2100b4be8a80d08675f041fbc719acf3b39dedc85dc92b3035868cb2daa8
f05db196a1003100
```

The breakdown of the payload is shown in [Appendix C.1](#).

[A.2.](#) enroll / reenroll

During the (re-)enroll exchange the EST-coaps client uses a CSR (Content-Format 286) request in the POST request payload. The Accept option tells the server that the client is expecting Content-Format

281 (PKCS#7) in the response. As shown in [Appendix C.2](#), the CSR contains a ChallengePassword which is used for PoP linking ([Section 4](#)).

```
POST [2001:db8::2:321]:61616/est/sen
(Token: 0x45)
(Accept: 281)
(Content-Format: 286)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
3082018b30820131020100305c310b3009060355040613025553310b3009
06035504080c024341310b300906035504070c024c413114301206035504
0a0c0b6578616d706c6520496e63310c300a060355040b0c03496f54310f
300d060355040513065774313233343059301306072a8648ce3d02010608
2a8648ce3d03010703420004c8b421f11c25e47e3ac57123bf2d9fdc494f
028bc351cc80c03f150bf50cff958d75419d81a6a245dffae790be95cf75
f602f9152618f816a2b23b5638e59fd9a073303406092a864886f70d0109
0731270c2576437630292a264a4b4a3bc3a2c280c2992f3e3c2e2c3d6b6e
7634332323403d204e787e60303b06092a864886f70d01090e312e302c30
2a0603551d1104233021a01f06082b06010505070804a013301106092b06
010401b43b0a01040401020304300a06082a8648ce3d0403020348003045
02210092563a546463bd9ecff170d0fd1f2ef0d3d012160e5ee90cffedab
ec9b9a38920220179f10a3436109051abad17590a09bc87c4dce5453a6fc
1135a1e84eed754377
```

After verification of the CSR by the server, a 2.04 Changed response with the issued certificate will be returned to the client.

2.04 Changed

(Token: 0x45)

(Content-Format: 281)

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
3082026e06092a864886f70d010702a082025f3082025b0201013100300b
06092a864886f70d010701a08202413082023d308201e2a0030201020208
7e7661d7b54e4632300a06082a8648ce3d040302305d310b300906035504
0613025553310b300906035504080c02434131143012060355040a0c0b45
78616d706c6520496e6331163014060355040b0c0d636572746966696361
74696f6e3113301106035504030c0a3830322e3141522043413020170d31
39303133313131323931365a180f39393939313233313233353935395a30
5c310b3009060355040613025553310b300906035504080c024341310b30
0906035504070c024c4131143012060355040a0c0b6578616d706c652049
6e63310c300a060355040b0c03496f54310f300d06035504051306577431
3233343059301306072a8648ce3d020106082a8648ce3d03010703420004
c8b421f11c25e47e3ac57123bf2d9fdc494f028bc351cc80c03f150bf50c
ff958d75419d81a6a245dffae790be95cf75f602f9152618f816a2b23b56
38e59fd9a3818a30818730090603551d1304023000301d0603551d0e0416
041496600d8716bf7fd0e752d0ac760777ad665d02a0301f0603551d2304
183016801468d16551f951bfc82a431d0d9f08bc2d205b1160300e060355
1d0f0101ff0404030205a0302a0603551d1104233021a01f06082b060105
05070804a013301106092b06010401b43b0a01040401020304300a06082a
8648ce3d0403020349003046022100c0d81996d2507d693f3c48eaa5ee94
91bda6db214099d98117c63b361374cd86022100a774989f4c321a5cf25d
832a4d336a08ad67df20f1506421188a0ade6d349236a1003100
```

The breakdown of the request and response is shown in [Appendix C.2](#).

[A.3.](#) serverkeygen

In a serverkeygen exchange the CoAP POST request looks like

POST 192.0.2.1:8085/est/skg
(Token: 0xa5)
(Accept: 62)
(Content-Format: 286)

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

3081d03078020100301631143012060355040a0c0b736b67206578616d70
6c653059301306072a8648ce3d020106082a8648ce3d03010703420004c8
b421f11c25e47e3ac57123bf2d9fdc494f028bc351cc80c03f150bf50cff
958d75419d81a6a245dffae790be95cf75f602f9152618f816a2b23b5638
e59fd9a000300a06082a8648ce3d040302034800304502207c553981b1fe
349249d8a3f50a0346336b7dfaa099cf74e1ec7a37a0a760485902210084
79295398774b2ff8e7e82abb0c17eaf344a5088fa69fd63ee611850c34b
0a

The response would follow [[I-D.ietf-core-multipart-ct](#)] and could look like

2.04 Changed

(Token: 0xa5)

(Content-Format: 62)

[The hexadecimal representations below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```

84                                     # array(4)
19 011C                             # unsigned(284)
58 8A                               # bytes(138)
308187020100301306072a8648ce3d020106082a8648ce3d030107046d30
6b020101042061336a86ac6e7af4a96f632830ad4e6aa0837679206094d7
679a01ca8c6f0c37a14403420004c8b421f11c25e47e3ac57123bf2d9fdc
494f028bc351cc80c03f150bf50cff958d75419d81a6a245dffae790be95
cf75f602f9152618f816a2b23b5638e59fd9
19 0119                             # unsigned(281)
59 01D3                             # bytes(467)
308201cf06092a864886f70d010702a08201c0308201bc0201013100300b
06092a864886f70d010701a08201a23082019e30820144a0030201020209
00b3313e8f3fc9538e300a06082a8648ce3d040302301631143012060355
040a0c0b736b67206578616d706c65301e170d3139303930343037343430
335a170d3339303833303037343430335a301631143012060355040a0c0b
736b67206578616d706c653059301306072a8648ce3d020106082a8648ce
3d03010703420004c8b421f11c25e47e3ac57123bf2d9fdc494f028bc351
cc80c03f150bf50cff958d75419d81a6a245dffae790be95cf75f602f915
2618f816a2b23b5638e59fd9a37b307930090603551d1304023000302c06
096086480186f842010d041f161d4f70656e53534c2047656e6572617465
64204365727469666963617465301d0603551d0e0416041496600d8716bf
7fd0e752d0ac760777ad665d02a0301f0603551d2304183016801496600d
8716bf7fd0e752d0ac760777ad665d02a0300a06082a8648ce3d04030203
48003045022100e95bfa25a08976652246f2d96143da39fce0dc4c9b26b9
cce1f24164cc2b12b602201351fd8eea65764e3459d324e4345ff5b2a915
38c04976111796b3698bf6379ca1003100

```

The private key in the response above is without CMS EnvelopedData and has no additional encryption beyond DTLS ([Section 5.8](#)).

The breakdown of the request and response is shown in [Appendix C.3](#)

[A.4. csrattrs](#)

Below is a csrattrs exchange

REQ:

GET example.com:61616/est/att

RES:

2.05 Content

(Content-Format: 285)

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
307c06072b06010101011630220603883701311b131950617273652053455
420617320322e3939392e31206461746106092a864886f70d010907302c06
0388370231250603883703060388370413195061727365205345542061732
0322e3939392e32206461746106092b240303020801010b06096086480165
03040202
```

A 2.05 Content response should contain attributes which are relevant for the authenticated client. This example is copied from Section A.2 in [\[RFC7030\]](#), where the base64 representation is replaced with a hexadecimal representation of the equivalent binary format. The EST-coaps server returns attributes that the client can ignore if they are unknown to him.

[Appendix B](#). EST-coaps Block message examples

Two examples are presented in this section:

1. a cacerts exchange shows the use of Block2 and the block headers
2. an enroll exchange shows the Block1 and Block2 size negotiation for request and response payloads.

The payloads are shown unencrypted. In practice the message contents would be binary formatted and transferred over an encrypted DTLS tunnel. The corresponding CoAP headers are only shown in [Appendix B.1](#). Creating CoAP headers is assumed to be generally known.

[B.1](#). cacerts

This section provides a detailed example of the messages using DTLS and BLOCK option Block2. The example block length is taken as 64 which gives an SZX value of 2.

The following is an example of a cacerts exchange over DTLS. The content length of the cacerts response in [appendix A.1 of \[RFC7030\]](#) contains 639 bytes in binary in this example. The CoAP message adds

around 10 bytes in this exmple, the DTLS record around 29 bytes. To avoid IP fragmentation, the CoAP Block Option is used and an MTU of 127 is assumed to stay within one IEEE 802.15.4 packet. To stay below the MTU of 127, the payload is split in 9 packets with a payload of 64 bytes each, followed by a last tenth packet of 63 bytes. The client sends an IPv6 packet containing a UDP datagram with DTLS record protection that encapsulates a CoAP request 10 times (one fragment of the request per block). The server returns an IPv6 packet containing a UDP datagram with the DTLS record that encapsulates the CoAP response. The CoAP request-response exchange with block option is shown below. Block Option is shown in a decomposed way (block-option:NUM/M/size) indicating the kind of Block Option (2 in this case) followed by a colon, and then the block number (NUM), the more bit (M = 0 in Block2 response means it is last block), and block size with exponent ($2^{*(SZX+4)}$) separated by slashes. The Length 64 is used with SZX=2. The CoAP Request is sent confirmable (CON) and the Content-Format of the response, even though not shown, is 281 (application/pkcs7-mime; smime-type=certs-only). The transfer of the 10 blocks with partially filled block NUM=9 is shown below

```
GET example.com:9085/est/crts (2:0/0/64) -->
    <-- (2:0/1/64) 2.05 Content
GET example.com:9085/est/crts (2:1/0/64) -->
    <-- (2:1/1/64) 2.05 Content
        |
        |
        |
GET example.com:9085/est/crts (2:9/0/64) -->
    <-- (2:9/0/64) 2.05 Content
```

The header of the GET request looks like


```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.1 GET)
Token = 0x9a    (client generated)
Options
Option (Uri-Host)
  Option Delta = 0x3  (option# 3)
  Option Length = 0xB
  Option Value = "example.com"
Option (Uri-Port)
  Option Delta = 0x4  (option# 3+4=7)
  Option Length = 0x2
  Option Value = 9085
Option (Uri-Path)
  Option Delta = 0x4  (option# 7+4=11)
  Option Length = 0x3
  Option Value = "est"
Option (Uri-Path)Uri-Path)
  Option Delta = 0x0  (option# 11+0=11)
  Option Length = 0x4
  Option Value = "crtts"
Option (Accept)
  Option Delta = 0x6  (option# 11+6=17)
  Option Length = 0x2
  Option Value = 281
Payload = [Empty]
```

The Uri-Host and Uri-Port Options can be omitted if they coincide with the transport protocol destination address and port respectively. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly.

For further detailing the CoAP headers, the first two and the last blocks are written out below. The header of the first Block2 response looks like


```
Ver = 1
T = 2 (ACK)
Code = 0x45 (2.05 Content)
Token = 0x9a      (copied from request by server)
Options
  Option
    Option Delta = 0xC (option# 12 Content-Format)
    Option Length = 0x2
    Option Value = 281
  Option
    Option Delta = 0xB (option# 12+11=23 Block2)
    Option Length = 0x1
    Option Value = 0x0A (block#=0, M=1, SZX=2)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
Payload =
3082027b06092a864886f70d010702a082026c308202680201013100300b
06092a864886f70d010701a082024e3082024a308201f0a0030201020209
009189bc
```

The second Block2:

```
Ver = 1
T = 2 (means ACK)
Code = 0x45 (2.05 Content)
Token = 0x9a      (copied from request by server)
Options
  Option
    Option Delta = 0xC (option# 12 Content-Format)
    Option Length = 0x2
    Option Value = 281
  Option
    Option Delta = 0xB (option 12+11=23 Block2)
    Option Length = 0x1
    Option Value = 0x1A (block#=1, M=1, SZX=2)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
Payload =
df9c99244b300a06082a8648ce3d0403023067310b300906035504061302
5553310b300906035504080c024341310b300906035504070c024c413114
30120603
```


The 10th and final Block2:

```
Ver = 1
T = 2 (means ACK)
Code = 0x45      (2.05 Content)
Token = 0x9a     (copied from request by server)
Options
  Option
    Option Delta = 0xC (option# 12 Content-Format)
    Option Length = 0x2
    Option Value = 281
  Option
    Option Delta = 0xB (option# 12+11=23 Block2 )
    Option Length = 0x1
    Option Value = 0x92 (block#=9, M=0, SZX=2)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
Payload =
2ec0b4af52d46f3b7ecc9687ddf267bcec368f7b7f1353272f022047a28a
e5c7306163b3c3834bab3c103f743070594c089aaa0ac870cd13b902caa1
003100
```

B.2. enroll / reenroll

In this example, the requested Block2 size of 256 bytes, required by the client, is transferred to the server in the very first request message. The block size $256 = (2^{SZX} * (SZX + 4))$ which gives $SZX = 4$. The notation for block numbering is the same as in [Appendix B.1](#). The header fields and the payload are omitted for brevity.


```

POST [2001:db8::2:1]:61616/est/sen (CON)(1:0/1/256) {CSR (frag# 1)} -->
    <-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR (frag# 2)} -->
    <-- (ACK) (1:1/1/256) (2.31 Continue)
    .
    .
    .
POST [2001:db8::2:1]:61616/est/sen (CON)(1:N1/0/256){CSR(frag# N1+1)}-->
    |
    .....Immediate response .....
    |
    <-- (ACK) (1:N1/0/256)(2:0/1/256)(2.04 Changed){Cert resp (frag# 1)}
POST [2001:db8::2:1]:61616/est/sen (CON)(2:1/0/256) -->
    <-- (ACK) (2:1/1/256)(2.04 Changed) {Cert resp (frag# 2)}
    .
    .
    .
POST [2001:db8::2:321]:61616/est/sen (CON)(2:N2/0/256) -->
    <-- (ACK) (2:N2/0/256) (2.04 Changed) {Cert resp (frag# N2+1)}

```

Figure 5: EST-COAP enrollment with multiple blocks

N1+1 blocks have been transferred from client to the server and N2+1 blocks have been transferred from server to client.

[Appendix C](#). Message content breakdown

This appendix presents the breakdown of the hexadecimal dumps of the binary payloads shown in [Appendix A](#).

[C.1](#). cacerts

The breakdown of cacerts response containing one root CA certificate is

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 831953162763987486 (0xb8bb0fe604f6a1e)

Signature Algorithm: ecdsa-with-SHA256

Issuer: C=US, ST=CA, L=LA, O=Example Inc,
OU=certification, CN=Root CA

Validity

Not Before: Jan 31 11:27:03 2019 GMT

Not After : Jan 26 11:27:03 2039 GMT

Subject: C=US, ST=CA, L=LA, O=Example Inc,
OU=certification, CN=Root CA

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:0c:1b:1e:82:ba:8c:c7:26:80:97:3f:97:ed:b8:
a0:c7:2a:b0:d4:05:f0:5d:4f:e2:9b:99:7a:14:cc:
ce:89:00:83:13:d0:96:66:b6:ce:37:5c:59:5f:cc:
8e:37:f8:e4:35:44:97:01:1b:e9:0e:56:79:4b:d9:
1a:d9:51:ab:45

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Subject Key Identifier:

1D:F1:20:89:44:D7:7B:5F:1D:9D:CB:51:EE:24:4A:52:3F:3E:F5:DE

X509v3 Authority Key Identifier:

keyid:

1D:F1:20:89:44:D7:7B:5F:1D:9D:CB:51:EE:24:4A:52:3F:3E:F5:DE

X509v3 Basic Constraints: critical

CA:TRUE

X509v3 Key Usage: critical

Certificate Sign, CRL Sign

X509v3 Subject Alternative Name:

email:certify@example.com

Signature Algorithm: ecdsa-with-SHA256

30:45:02:20:2b:89:1d:d4:11:d0:7a:6d:6f:62:19:47:63:5b:
a4:c4:31:65:29:6b:3f:63:37:26:f0:2e:51:ec:f4:64:bd:40:
02:21:00:b4:be:8a:80:d0:86:75:f0:41:fb:c7:19:ac:f3:b3:
9d:ed:c8:5d:c9:2b:30:35:86:8c:b2:da:a8:f0:5d:b1:96

[C.2.](#) enroll / reenroll

The breakdown of the enrollment request is

Certificate Request:

Data:

Version: 0 (0x0)

Subject: C=US, ST=CA, L=LA, O=example Inc,
OU=IoT/serialNumber=Wt1234

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:

9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:

0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:

be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:

56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

Attributes:

challengePassword: <256-bit PoP linking value>

Requested Extensions:

X509v3 Subject Alternative Name:

othername:<unsupported>

Signature Algorithm: ecdsa-with-SHA256

30:45:02:21:00:92:56:3a:54:64:63:bd:9e:cf:f1:70:d0:fd:

1f:2e:f0:d3:d0:12:16:0e:5e:e9:0c:ff:ed:ab:ec:9b:9a:38:

92:02:20:17:9f:10:a3:43:61:09:05:1a:ba:d1:75:90:a0:9b:

c8:7c:4d:ce:54:53:a6:fc:11:35:a1:e8:4e:ed:75:43:77

The CSR contains a ChallengePassword which is used for PoP linking ([Section 4](#)). The CSR also contains an id-on-hardwareModuleName hardware identifier to customize the returned certificate to the requesting device (See [[RFC7299](#)] and [[I-D.moskowitz-ecdsa-pki](#)]).

The breakdown of the issued certificate is

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 9112578475118446130 (0x7e7661d7b54e4632)

Signature Algorithm: ecdsa-with-SHA256

Issuer: C=US, ST=CA, O=Example Inc,
OU=certification, CN=802.1AR CA

Validity

Not Before: Jan 31 11:29:16 2019 GMT

Not After : Dec 31 23:59:59 9999 GMT

Subject: C=US, ST=CA, L=LA, O=example Inc,
OU=IoT/serialNumber=Wt1234

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:
9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:
0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:
be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:
56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

X509v3 Subject Key Identifier:

96:60:0D:87:16:BF:7F:D0:E7:52:D0:AC:76:07:77:AD:66:5D:02:A0

X509v3 Authority Key Identifier:

keyid:

68:D1:65:51:F9:51:BF:C8:2A:43:1D:0D:9F:08:BC:2D:20:5B:11:60

X509v3 Key Usage: critical

Digital Signature, Key Encipherment

X509v3 Subject Alternative Name:

othername:<unsupported>

Signature Algorithm: ecdsa-with-SHA256

30:46:02:21:00:c0:d8:19:96:d2:50:7d:69:3f:3c:48:ea:a5:
ee:94:91:bd:a6:db:21:40:99:d9:81:17:c6:3b:36:13:74:cd:
86:02:21:00:a7:74:98:9f:4c:32:1a:5c:f2:5d:83:2a:4d:33:
6a:08:ad:67:df:20:f1:50:64:21:18:8a:0a:de:6d:34:92:36**C.3. serverkeygen**

The following is the breakdown of the server-side key generation request.

Certificate Request:

Data:

Version: 0 (0x0)

Subject: 0=skg example

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:

9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:

0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:

be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:

56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

Attributes:

a0:00

Signature Algorithm: ecdsa-with-SHA256

30:45:02:20:7c:55:39:81:b1:fe:34:92:49:d8:a3:f5:0a:03:

46:33:6b:7d:fa:a0:99:cf:74:e1:ec:7a:37:a0:a7:60:48:59:

02:21:00:84:79:29:53:98:77:4b:2f:f8:e7:e8:2a:bb:0c:17:

ea:ef:34:4a:50:88:fa:69:fd:63:ee:61:18:50:c3:4b:0a

Following is the breakdown of the private key content of the server-side key generation response.

Private-Key: (256 bit)

priv:

61:33:6a:86:ac:6e:7a:f4:a9:6f:63:28:30:ad:4e:

6a:a0:83:76:79:20:60:94:d7:67:9a:01:ca:8c:6f:

0c:37

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:

9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:

0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:

be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:

56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

The following is the breakdown of the certificate in the server-side key generation response payload.

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

b3:31:3e:8f:3f:c9:53:8e

Signature Algorithm: ecdsa-with-SHA256

Issuer: O=skg example

Validity

Not Before: Sep 4 07:44:03 2019 GMT

Not After : Aug 30 07:44:03 2039 GMT

Subject: O=skg example

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:

9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:

0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:

be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:

56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

96:60:0D:87:16:BF:7F:D0:E7:52:D0:AC:76:07:77:AD:66:5D:02:A0

X509v3 Authority Key Identifier:

keyid:

96:60:0D:87:16:BF:7F:D0:E7:52:D0:AC:76:07:77:AD:66:5D:02:A0

Signature Algorithm: ecdsa-with-SHA256

30:45:02:21:00:e9:5b:fa:25:a0:89:76:65:22:46:f2:d9:61:

43:da:39:fc:e0:dc:4c:9b:26:b9:cc:e1:f2:41:64:cc:2b:12:

b6:02:20:13:51:fd:8e:ea:65:76:4e:34:59:d3:24:e4:34:5f:

f5:b2:a9:15:38:c0:49:76:11:17:96:b3:69:8b:f6:37:9c

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

URI: <http://www.sandelman.ca/>

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se

