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 August 10, 2019.

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.
Table of Contents

1. Change Log .................................................. 3
2. Introduction ................................................ 5
3. Terminology .................................................. 6
4. Conformance to RFC7925 profiles .......................... 6
5. Protocol Design .............................................. 7
   5.1. Discovery and URIs .................................... 8
   5.2. Mandatory/optional EST Functions .................... 10
   5.3. Payload formats ....................................... 10
   5.4. Message Bindings ..................................... 12
   5.5. CoAP response codes .................................. 13
   5.6. Message fragmentation ................................. 13
   5.7. Delayed Responses ................................... 14
   5.8. Server-side Key Generation ......................... 16
6. DTLS Transport Protocol ................................. 18
7. HTTPS-CoAPS Registrar ................................. 19
8. Parameters .................................................. 21
9. Deployment limitations ................................... 21
10. IANA Considerations ..................................... 22
   10.1. Content-Format Registry ............................... 22
   10.2. Resource Type registry ............................... 22
11. Security Considerations ................................... 23
   11.1. EST server considerations ........................... 23
   11.2. HTTPS-CoAPS Registrar considerations ............... 25
12. Contributors .............................................. 25
13. Acknowledgements ......................................... 26
14. References ................................................ 26
   14.1. Normative References ................................ 26
   14.2. Informative References ............................... 28
Appendix A. EST messages to EST-coaps ....................... 30
   A.1. cacerts ........................................... 31
   A.2. enroll / reenroll .................................. 33
   A.3. serverkeygen ....................................... 35
   A.4. csrattrs ........................................... 37
Appendix B. EST-coaps Block message examples ............... 38
   B.1. cacerts ........................................... 38
   B.2. enroll / reenroll .................................. 42
Appendix C. Message content breakdown ....................... 43
   C.1. cacerts ........................................... 43
   C.2. enroll / reenroll .................................. 45
C.3. serverkeygen ........................................ 46
Authors’ Addresses ........................................ 48

1. Change Log

EDNOTE: Remove this section before publication

-08

  added application/pkix-cert Content-Format TBD287.

  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

  Persistenc 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 7. 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], CoAP [RFC7252] and UDP instead of TLS [RFC8446], HTTP [RFC7230] and TCP.

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. Conformance to RFC7925 profiles

This section describes how EST-coaps fits into 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 for 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.

As per Sections 3.3 and 4.4 of [RFC7925], the mandatory cipher suite for DTLS in EST-coaps is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. Curve secp256r1 MUST be supported [RFC8422]; this curve is equivalent to the NIST P-256 curve. Crypto agility is important, and the recommendations in [RFC7925] section 4.4 and any updates to RFC7925 concerning Curve25519 and other CFRG curves also apply.

DTLS1.2 implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC8422]. Uncompressed point format MUST also be supported. [RFC6090] is a summary of the ECC algorithms. 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.

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 from its manufacturer 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
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.

- a previously installed certificate (e.g., manufacturer IDevID [ieee802.1ar] or a certificate issued by some other party); the server is expected to trust that certificate. 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 11).

5. Protocol Design

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to avoid (excessive) fragmentation of UDP datagrams. The use of Blocks for the transfer of larger EST messages is specified in Section 5.6. Figure 1 below shows the layered EST-coaps architecture.

```
+------------------------------------------------+
|    EST request/response messages                |
| +------------------------------------------------+
|    CoAP for message transfer and signaling     |
| +------------------------------------------------+
|    DTLS for transport security                 |
| +------------------------------------------------+
|    UDP for transport                            |
+------------------------------------------------+
```

Figure 1: EST-coaps protocol layers

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

- CA certificate retrieval, needed to receive the complete set of CA certificates.

- Simple enroll and reenroll, for a CA to sign public client-identity key.

- Certificate Signing Request (CSR) Attributes messages that informs the client of the fields to include in generated CSR.

- Server-side key generation messages to provide a private client-identity key when the client choses so.
5.1. Discovery and URIs

EST-coaps is targeted for low-resource networks with small packets. 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://est-coaps.example.ietf.org:<port>/.well-known/est/<short-est>
coaps://est-coaps.example.ietf.org:<port>/.well-known/est/
ArbitraryLabel/<short-est>

The short-est strings are defined in Table 1. The ArbitraryLabel path-segment, if used, SHOULD be of the shortest length possible (Sections 3.1 and 3.2.2 of [RFC7030]). Arbitrary Labels are usually defined and used by EST CAs in order to route client requests to the appropriate certificate profile.

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

coaps://est-coaps.example.ietf.org:<port>/<root-resource>/<short-est>
coaps://est-coaps.example.ietf.org:<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.

<table>
<thead>
<tr>
<th>EST</th>
<th>EST-coaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cacerts</td>
<td>/crts</td>
</tr>
<tr>
<td>/simpleenroll</td>
<td>/sen</td>
</tr>
<tr>
<td>/simplereenroll</td>
<td>/sren</td>
</tr>
<tr>
<td>/csrattrs</td>
<td>/att</td>
</tr>
<tr>
<td>/serverkeygen</td>
<td>/skg</td>
</tr>
</tbody>
</table>

Table 1: Short EST-coaps URI path

Clients and servers MUST support the short resource URIs. The corresponding longer URIs from [RFC7030] MAY be supported.

In the context of CoAP, the presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "ace.est" [RFC6690]. Upon success, the return payload will contain the root resource of the EST resources. The example below shows the
discovery 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

```plaintext
<coap://[2001:db8:3::123]:61617/est>;rt="ace.est";
anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/crts>;rt="ace.est.crts";
ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/sen>;rt="ace.est.sen";
ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/sren>;rt="ace.est.sren";
ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/att>;rt="ace.est.att";
ct="285";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/skg>;rt="ace.est.skg";
ct="62 280 284 281 TBD287";anchor="coap://[2001:db8:3::123]:61617"
```

The first line of the discovery response above MUST be included. The five consecutive lines after the first MAY be included. The return of the content types allows the client to choose the most appropriate one.

Port numbers, not returned in the example, are assumed to be the default numbers 5683 and 5684 for CoAP and CoAPS respectively (Sections 12.6 and 12.7 of [RFC7252]). Discoverable port numbers MAY be returned in the <href> of the payload [I-D.ietf-core-resource-directory]. An example response payload for non-default CoAPS server port 61617 follows below. Linefeeds were included only for readability.

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

RES: 2.05 Content

```plaintext
<coap://[2001:db8:3::123]:61617/est>;rt="ace.est";
anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/crts>;rt="ace.est.crts";
ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/sen>;rt="ace.est.sen";
ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/sren>;rt="ace.est.sren";
ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/att>;rt="ace.est.att";
ct="285";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/skg>;rt="ace.est.skg";
ct="62 280 284 281 TBD287";anchor="coap://[2001:db8:3::123]:61617"
```

The server MUST support the default /.well-known/est server root resource and port 5684. Resource discovery is necessary when the IP address of the server is unknown to the client. Resource discovery SHOULD be employed when non-default URIs (like /est or /est/ArbitraryLabel) or ports are supported by the server, when the client is unaware of what EST-coaps resources are available or if the client
considers sending two Uri-Path Options to convey the resource is wasteful.

It is up to the implementation to choose its root resource; throughout this document the example root resource /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.

<table>
<thead>
<tr>
<th>EST Functions</th>
<th>EST-coaps implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cacerts</td>
<td>MUST</td>
</tr>
<tr>
<td>/simpleenroll</td>
<td>MUST</td>
</tr>
<tr>
<td>/simplereenroll</td>
<td>MUST</td>
</tr>
<tr>
<td>/fullcmc</td>
<td>Not specified</td>
</tr>
<tr>
<td>/serverkeygen</td>
<td>OPTIONAL</td>
</tr>
<tr>
<td>/csrattrs</td>
<td>OPTIONAL</td>
</tr>
</tbody>
</table>

Table 2: List of EST-coaps functions

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

5.3. Payload formats

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 in EST-coaps 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 10.1. 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 /crts, /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 is expected to use an 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. If the preferred
Content-Format cannot be returned, the server MUST send a 4.06 (Not
Acceptable) response, unless another error code takes precedence for
the response [RFC7252].

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

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) 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.

*application/multipart-core*

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].

The collection is encoded as a CBOR array [RFC7049] with an even
number of elements. The second, fourth, sixth, etc. element is a
binary string containing a representation. The first, third, fifth,
etc. element is an unsigned integer specifying the Content-Format
identifier of the consecutive representation. For example, a
collection containing two representations in response to an EST-coaps
server-side key generation 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
Multipart /skg response serialization

When the returned certificate is a single X.509 certificate (not a PKCS#7 container) the Content-Format identifier is 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 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:

- All EST-coaps messages expect a response from the server, thus the client MUST send the requests over confirmable CON CoAP messages.

- The Ver, TKL, Token, and Message ID values of the CoAP header are not affected by EST.

- The CoAP Options used are Uri-Host, Uri-Path, Uri-Port, Content-Format, Accept and Location-Path. These CoAP Options are used to communicate the HTTP fields specified in the EST REST messages. The Uri-Host and Uri-Port Options are optional. They are usually omitted as the DTLS destination and port are sufficient. 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. Alternatively, if a UDP port to a server is blocked, someone could send the DTLS packets to a known open port on the server and use the Uri-Port to convey the intended port he is attempting to reach.

- 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. Every time the HTTP response code 200 is specified in [RFC7030] in response to a GET request (/cacerts, /csrattrs), in EST-coaps the equivalent CoAP response code 2.05 or 2.03 MUST be used. Similarly, 2.01, 2.02 or 2.04 MUST be used in response to EST POST requests (/simpleenroll, /simploenroll, /serverkeygen).

Response code HTTP 202 Retry-After that existed in EST has no equivalent in CoAP. Retry-After is used in EST for delayed server responses. Section 5.7 specifies how EST-coaps handles delayed messages.

EST makes use of HTTP 204 and 404 responses when a resource is not available for the client. The equivalent CoAP codes to use in an EST-coaps responses are 2.04 and 4.04. Additionally, EST’s HTTP 401 error translates to 4.01 in EST-coaps. Other EST HTTP error messages are 400, 423 and 503. Their equivalent CoAP errors are 4.00, 4.03 and 5.03 respectively. In case a CoAP Option is unrecognized and critical, the server is expected to return a 4.02 (Bad Option). Moreover, if the Content-Format requested in the client Accept Option, is not supported the server MUST return a 4.06 (Not Acceptable), unless another error code takes precedence for the response.

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 SHOULD attempt 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. The EST-coaps client and server MUST support Block2. Block1 MUST be supported for EST-coaps enrollment requests that exceed 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 server after a short delay. If the certificate response is large, the server will need more than one Block2 blocks 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 with an ACK. 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 Uri-Path and acknowledges the Block2 blocks sent up to that point.
POST [2001:db8::2:1]:61616/est/sen (CON)(1:0/1/256) {CSR req} -->
  <-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR req} -->
  <-- (ACK) (1:1/1/256) (2.31 Continue)

...... short delay before certificate is ready ......

POST [2001:db8::2:1]:61616/est/sen (CON)(1:N1/0/256) {CSR req} -->
  <-- (0.00 empty ACK)

POST [2001:db8::2:1]:61616/est/sen (CON)(1:N1/0/256) {CSR req} -->
  <-- (CON) (1:N1/0/256) (2:0/1/256) (2.04 Changed) {Cert resp}
  (ACK) -->

POST [2001:db8::2:1]:61616/est/sen (CON)(2:1/0/128) -->
  <-- (ACK) (2:1/1/128) (2.04 Changed) {Cert resp}

POST [2001:db8::2:1]:61616/est/sen (CON)(2:N2/0/128) -->
  <-- (ACK) (2:N2/0/128) (2.04 Changed) {Cert resp}

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), the server 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 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 he receives the same 5.03 response and retransmits the enrollment request until he receives a certificate in a fragmented 2.01 response. Note that the server asks 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 req} -->
      <-- (ACK) (1:0/1/256) (2.31 Continue)

POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR req} -->
      <-- (ACK) (1:1/1/256) (2.31 Continue)

POST [2001:db8::2:1]:61616/est/sen (CON)(1:N1/0/256){CSR req} -->
      <-- (ACK) (1:N1/0/256) (2:0/0/128)(5.03 Service Unavailable)
          (Max-Age)

Client tries one or more times after Max-Age with identical payload

POST [2001:db8::2:1]:61616/est/sen (CON)(1:N1/0/256){CSR req} -->
      <-- (ACK) (1:N1/0/256) (2:0/1/128) (2.01 Created){Cert resp}

POST [2001:db8::2:1]:61616/est/sen (CON)(2:1/0/128)           -->
      <-- (ACK) (2:1/1/128) (2.01 Created) {Cert resp}

POST [2001:db8::2:1]:61616/est/sen (CON)(2:N2/0/128)          -->
      <-- (ACK) (2:N2/0/128) (2.01 Created) {Cert resp}

Figure 3: EST-COAP enrollment with long wait

5.8. Server-side Key Generation

Constrained devices sometimes do not have the necessary hardware to
generate statistically random numbers for private keys and DTLS
ephemeral keys. Past experience has also shown that low-resource
endpoints sometimes generate numbers which could allow someone to
decrypt the communication or guess the private key and impersonate as
the device [PsQs] [RSAorig]. Additionally, random number key
generation is costly, thus energy draining. Even though the random
numbers that constitute the identity/cert do not get generated often,
an endpoint may not want to spend time and energy generating
keypairs, and just ask for one from the server.

In these scenarios, server-side key generation can be used. 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
SHOULD treat 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 needs to communicate to the server the Content-Format of the application/multipart-core elements. The key Content-Format requested by the client is depicted in the PKCS#10 request. If the request contains SMIMECapabilities the client is expecting Content-Format 280. Otherwise he expects a PKCS#8 key in Content-Format 284. The client expresses the preferred certificate Content-Format in his /skg request by using an Accept Option. The Accept Option is 281 when preferring a certificate in a PKCS#7 container or TBD287 when preferring a single X.509 certificate.

[RFC7030] provides two methods, symmetric and asymmetric, to optionally encrypt the generated key. The methods are signaled by the client by using the relevant attributes (SMIMECapabilities and DecryptKeyIdentifier or AsymmetricDecryptKeyIdentifier) in the CSR request. The symmetric key or the asymmetric keypair establishment method is out of scope of the specification.

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 certificate part exactly matches the response from an enrollment response. 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 is signed by the party that generated the private key, which may be the EST server or the EST CA. The SignedData is further protected by placing it inside of a CMS EnvelopedData 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 needs to be provided 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).

6. DTLS Transport Protocol

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

CoAP was designed to avoid 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 DTLS handshake is authenticated by using certificates. EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in Section 3 of [RFC7030].

CoAP and DTLS can provide proof-of-identity for EST-coaps clients and servers with simple PKI messages as described in Section 3.1 of [RFC5272]. Moreover, channel-binding information for linking proof-of-identity with connection-based proof-of-possession is OPTIONAL for EST-coaps. 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 11.1.

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.

7. 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.

```
+-------+                         +-----------------+
| CA    |                         |.-----------------.|
|       |                         ||                          ||
|-------|  HTTP               ||                          ||
| EST   |<------>|EST-coaps-to-HTTPS|<------>| EST Client|
|Server | over TLS |   Registrar     |          '-----------'
|       |          '-----------------'                         ||
|       |                          ||'--------------------------'|
|       |                          ||'----------------------------'
```

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 clients and 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 and was used to establish the connection by the client as explained in Section 6. 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. In these cases, the Registrar MUST support random number generation using proper entropy. Such 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.

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 10.1. Additionally, a conversion from CBOR major type 2 to Base64 encoding MUST take place at the Registrar when server-side key generation is supported. If CMS end-to-end encryption is employed for the private key, the encrypted CMS EnvelopedData blob MUST be converted to binary in 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.
For the discovery of the EST server by the EST client in the CoAP environment, the EST-coaps-to-HTTP Registrar MUST announce itself according to the rules in Section 5.1. The available actions of the Registrars MUST be announced with as many resource paths necessary.

8. 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 EST parameter values need to be tuned to the CoAP 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. A change in a server parameter MUST ensure the adjusted value is also available to all the endpoints with which these adjusted values are to be used to communicate.

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

- NSTART: Limit the number of simultaneous outstanding interactions that a client maintains to a given server. EST-coaps clients SHOULD use 1, which is the default. A EST-coaps client is not expected to interact with more than one servers at the same time.

- DEFAULT_LEISURE: This setting is only relevant in multicast scenarios, outside the scope of EST-coaps.

- PROBING_RATE: A parameter which specifies the rate of re-sending non-confirmable messages. The EST messages are defined to be sent as CoAP confirmable messages, hence this setting is not applicable.

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.

9. 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 large 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 UDP/DTLS/CoAP. It is
up to the network designer to decide which devices execute the EST protocol and which do not.

10. IANA Considerations

10.1. Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry [COREparams] are specified in Table 3. These have been registered provisionally in the Expert Review range (0-255).

<table>
<thead>
<tr>
<th>HTTP Media-Type</th>
<th>ID</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/pkcs7-mime; smime-type=server-generated-key</td>
<td>280</td>
<td>[RFC7030] [I-D.ietf-lamps-rfc5751-bis]</td>
</tr>
<tr>
<td>application/pkcs7-mime; smime-type=certs-only</td>
<td>281</td>
<td>[I-D.ietf-lamps-rfc5751-bis]</td>
</tr>
<tr>
<td>application/pkcs7-mime; smime-type=CMC-request</td>
<td>282</td>
<td>[RFC5273] [I-D.ietf-lamps-rfc5751-bis]</td>
</tr>
<tr>
<td>application/pkcs7-mime; smime-type=CMC-response</td>
<td>283</td>
<td>[RFC5273] [I-D.ietf-lamps-rfc5751-bis]</td>
</tr>
<tr>
<td>application/pkcs8</td>
<td>284</td>
<td>[RFC5958] [I-D.ietf-lamps-rfc5751-bis]</td>
</tr>
<tr>
<td>application/csrattrs</td>
<td>285</td>
<td>[RFC7030] [RFC7231]</td>
</tr>
<tr>
<td>application/pkcs10</td>
<td>286</td>
<td>[RFC5967] [I-D.ietf-lamps-rfc5751-bis]</td>
</tr>
<tr>
<td>application/pkix-cert</td>
<td>TBD28</td>
<td>[RFC2585]</td>
</tr>
</tbody>
</table>

Table 3: New CoAP Content-Formats

The Content-Formats 281 to 286 have been the subject of an earlier temporary allocation. It is suggested that 287 is allocated to TBD287.

10.2. Resource Type registry

This memo registers a 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.

- rt="ace.est". This EST resource is used to query and return the supported EST resources of a CoAP server.
o rt="ace.est.crts". 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 CSR attributes.

o rt="ace.est.skg". This resource depicts the support of EST server-side key generation.

11. Security Considerations

11.1. 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.

Given that the client has only limited resources and may not be able to generate sufficiently random keys to encrypt its identity, it is possible that the client uses server generated private/public keys. The transport of these keys is inherently risky. Analysis SHOULD be done to establish whether server-side key generation enhances or decreases the probability of identity stealing.

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 6). A client that pipelines 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, 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. The attack was possible because of certain (D)TLS implementation imperfections. 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]). Such mechanism has not been standardized yet. Adopting in this document a channel binding value generated from an exporter would break backwards compatibility. Thus, in this specification we still depend in the tls-unique mechanism defined in [RFC5929], especially since the practicality of such an attack would not expose any messages exchanged with EST-coaps.

Regarding the Certificate Signing Request (CSR), a CA is expected to be able to enforce policies 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.
11.2. HTTPS-CoAPS Registrar considerations

The Registrar proposed in Section 7 must be deployed with care, and only when the recommended connections are impossible. When POP linking is used the Registrar terminating the TLS connection establishes a new one 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 client has verified this information when acting as an EST server.

The introduction of an EST-coaps-to-HTTP Registrar assumes the client can trust 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, he SHOULD confirm if the server is acting as an RA by the presence of the id-kp-cmcRA [RFC6402] EKU in the server certificate. If the server certificate does not include the EKU, it is RECOMMENDED that the client includes "Linking Identity and POP Information" (Section 6) in requests.

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 doing the right thing if it is generating the private key.

12. Contributors

Martin Furuhed 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.
13. 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 and Pete Beal.

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

Robert Moskowitz provided code to create the examples.

14. References

14.1. Normative References

[I-D.ietf-core-multipart-ct]

[I-D.ietf-tls-dtls13]


14.2. Informative References

[COREparams]

[I-D.ietf-core-resource-directory]

[I-D.ietf-lamps-rfc5751-bis]

[I-D.ietf-tls-dtls-connection-id]

[I-D.josefsson-sasl-tls-cb]

[I-D.moskowitz-ecdsa-pki]

[ieee802.15.4]
Institute of Electrical and Electronics Engineers, "IEEE Standard 802.15.4-2006", 2006.

[ieee802.1ar]
Institute of Electrical and Electronics Engineers, "IEEE 802.1AR Secure Device Identifier", December 2009.

[PsQs]


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]. The payloads are shown unencrypted. In practice the message content would be binary.
formatted and transferred over an encrypted DTLS tunnel. The hexadecimal representations in the examples below would NOT be transported in hex, but in binary. Hex is used for visualization purposes because a binary representation cannot be rendered well in text.

The certificate responses included in the examples contain Content-Format 281 (application/pkcs7). If the client had requested Content-Format TBD287 (application/pkix-cert) with an Accept Option, the server would respond a single DER binary certificate.

These examples assume that the resource discovery, returned a short base path of "/est".

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 coaps://est-coaps.example.ietf.org: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) [optional]
  Option Delta = 0x3  (option# 3)
  Option Length = 0x9
  Option Value = est-coaps.example.ietf.org
Option (Uri-Port) [optional]
  Option Delta = 0x4  (option# 3+4=7)
  Option Length = 0x4
  Option Value = 9085
Option (Uri-Path)
  Option Delta = 0x4  (option# 7+4=11)
  Option Length = 0x5
  Option Value = "est"
Option (Uri-Path)
  Option Delta = 0x0  (option# 11+0=11)
  Option Length = 0x6
  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 are optional. They are usually omitted as the DTLS destination and port are sufficient. 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. Alternatively, if a UDP port to a server is blocked, someone could send the DTLS packets to a known open port on the server and use the Uri-Port to convey the intended port he is attempting to reach.

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 =
3082027b06092a864886f70d010702a082026c30820268208201013100300b
06092a864886f70d010701a082024e3082024a308201fa0030201020209
009189b92f9c99244b30a06068a2a6484ce3d0403023067310b3009060355
04061302555310b300906035504080c024341310b300906035504070c02
4c41311430120635504a0c0b04578616d706c6520496e6331163014060355040b0c0d6365727469666963
617469666e3110300e0635504030c075266f74204341310b300906035504072a
864ce3d02010608a2a6484ce3d0301070342004814994082b6e8185f3df
53f5e0bee69897335200023dff78cd17a443ff8dd4098769c55652ac
2ccc75c4a50a7c7cdb7c22da6e685cca53829f4dbbf104ca3818430818
301d0603551d0e041604142495e816ed6f6ffcaaf356ce4adffe33cf492abb
a8301f063551d230418301680142495e816ed6f6ffcaaf356ce4adffe33cf
492abba8300f063551d130101ff04053030101ff300e063551d0f0101
ff0403020106301e063551d1104730158113636572746674666678
616d706c6526366fd300a0608a2a6484ce3d040302348003045022100da
e37c96f15c32ec0b4af52d4f6f3b7ecc9687dd27bce368f7b7f135327
2f022047a28ae5c7306163b3c3834bab3c103f743070594c089aa0ac870
cd13b902ca1003100

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 6).
POST [2001:db8::2:1]: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. ]

3082018b308201310100305a310b300906035504061302553210b3009
0603550408a024341310b300906035504070c024c413114301206035504
0a0c0b658616d706c6520496e63310c30a060355040b0c03496f54310f
300d60355040530365774313233343059301306072a8648ce3d02010608
2a8648ce3d03010703420004c8b421f11c25e47e3ac75123bf2d9f49f
028bc351cc80c03f150bf50caff958d75419d81a6a245ddfae790be95cf75
f602f9152618f816a2b23b5638e59fd9a073303406092a864886f70d0109
0731270c2576437630292a264a4b4a3bc3a2c280c2992f3e3c2e2c3d6b6e
763432323403d20a4787e60303b06092a864886f70d0109e312e302c30
2a0603551d110423021a01f06082b06010505070804a013301106092b06
010401b43ba0a1040401020304300a06082a8648ce3d0403020348003045
02210092563a54643bd9ecff170d0f1f2ef0d3d012160e5ee90ccfedab
e8e9a8392020179410a3436109051abad17590a09bc87c4dce5453a6fc
1135a1e84ed754377

After verification of the CSR by the server, a 2.01 Content response with the issued certificate will be returned to the client.
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.

```
3082026a06092a864886f70d010702a082025f3082025b0b201013100300b
06092a864886f70d010701a08202413082023d308201e2a0030201020208
7e76617b54e4632300a06082a8648ca3d040302305d310b300906035504
061302553310b300906035504080c024341311431012060355040a0c0b45
78616d706c6520496e633116310160435504030c0a3830322e31452204321070d31
393031333131323931365a180f393939393132333132333335393539a30
5c310b30090603550406120555310b300906035504080c024341310b30
0906035504070c024131143012060355040a0c0b6578616d706c652049
6e63310c300a060355040b0c03496f54310f300d06035504051306577431
3233343059301306072a8648ce3d020106082a8648ce3d03010703420004
c8b421f11c25e473e3ac57123bf2d9f4c494f028bc351cc8003f150bf50c
ff95875419d81a6a245dfbfae790be95cf75602f9152618f816a2b3b56
38e59fd9a3818a30818730090603551d1304023000301d0603551d0e0416
04149660d8716bf7fde0752d0ac7670777ad665d02a0301f0603551d2304
183016801468d16551f951bfc82a431d09f08bc2d205b1160300e060355
1d0f010ff004003205a0302a0603551d11042332021a01f06082b060105
0507080a013301106092b06010401b43b0a010401020304300a06082a
864ce3d04032034900304622100cd81996d2507d693fc48ea5ee94
91bd6db214099d98117c63b361374cd86022100a774989f4c321a5cf25d
83a4d336a08ad67df20f1506421188a0ade6349236a1003100
```

The breakdown of the request and response is shown in Appendix C.2.

As described in Section 5.7, if the server is not able to provide a response immediately, it sends an empty ACK with response code 5.03 (Service Unavailable) and the Max-Age Option. See Figure 3 for an example exchange.

### A.3. serverkeygen

In a serverkeygen exchange the CoAP POST request looks like
POST coaps://192.0.2.1:8085/est/skg
(Token: 0xa5)
(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. ]

3081cf3078020100301631143012060355040a0c0b736b67206578616d70
6c65305930130672a8648ce3d020106082a8648ce3d030107034200041b
b8c117896f98e4506c03d70efbe820d8e38ea97e9d65d52c8460c5852c5
1dd89a61370a2843760fc859799d78cd33f3c1846e304f1717f8123f1a28
4cc99fa000300a06082a8648ce3d0403020347003044020387cd4e9cf62
8d4af77f92ebed4890d9d141dca86cd2757d14cb59cfd6961802202f24
5e828c77754378b66660a4977f113cacdaa0cc7bad7d1474a7fd155d90d

The response would follow [I-D.ietf-core-multipart-ct] and could look like
2.01 Content
(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
19 011C   # array(4)
58 8A     # bytes(138)
3081702010030106072a8648ce3d020106082a8648ce3d030107046d30
6b02010104200b9a67785b65e07360b6d28cf1cd3f3925c0755799deea7
45372b01697bd8a6a144034200041bb8c1117896f98e4506c03d70efbe82
0d8e38ea97e9d65c8460c582c51dd89a61370a2843760fc859799d78
cd33f3c1846e304f1717f8123f1a284cc99f
19 0119   # unsigned(281)
59 01D3   # bytes(467)
308201cf06092a864886f70d010702a08201c0308201bc0201011300300b
06092a864886f70d010701a08201a23082019e30820143a0030201020208
126de8571518524b300a06082a8648ce3d04030230163114301206035504
0a00cb736b67206578616d706c65301e170d313930313039303835373038
5a170d33393031303403835373038a5a30163114301206035504a0c0b73
6b7206578616d706c653059301306072a8648ce3d020106082a8648ce3d
030107034200041bb8c1117896f98e4506c03d70ebbe820d8e38ea97e9d6
5d52c8460c582c51dd89a61370a2843760fc859799d78cd33f3c1846e30
4f1717f8123f1a284cc99fa37b307930090603551d1304023000302c0609
6084680186f842010d041f161d4f70656e53534c2047656e657261746564
204365727469666963617465501d0603551d0e0416041449be598dc8dbc
0dbc071c486b7774605cce621301f0603551d23041830168014494be598
dc8dbc0dbc071c486b7774605cce621300a06082a8648ce3d0403020349
0030460221004b167d0f9ad9202810e6bf6a290b8cfdfc9b9c9f2ac2c1
c8fc3a464f79f2c202210081d31ba142751a7b4a34fd1a01fcfb80716b9e
b53baada9e60b08f52429c0fa1003100

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 coaps://[2001:db8::2:1]: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. ]

307c06072b06010101011630220603883701311b131950617273652053455417320322e3939392e312061746106092a864886f70d010907302c06
0388370231250603883703063883704131950617273652053455420617320322e3939392e3220617461746106092b240303020801010b06096086480165
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 are assumed to be generally
known.

B.1. cacerts

This section provides a detailed example of the messages using DTLS
and BLOCK option Block2. The minimum PMTU is 1280 bytes, which is
the example value assumed for the DTLS datagram size. 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. The CoAP message adds around 10 bytes, the DTLS record 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 the UDP datagram with the DTLS record that encapsulates the CoAP request 10 times. The server returns an IPv6 packet containing the 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 to avoid IP fragmentation. 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 coaps://est-coaps.example.ietf.org:9085/est/crts (2:0/0/64) -->
  <-- (2:0/1/64) 2.05 Content
GET coaps://est-coaps.example.ietf.org:9085/est/crts (2:1/0/64) -->
  <-- (2:1/1/64) 2.05 Content
  
GET coaps://est-coaps.example.ietf.org: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) [optional]
    Option Delta = 0x3 (option# 3)
    Option Length = 0x9
    Option Value = est-coaps.example.ietf.org
  Option (Uri-Port) [optional]
    Option Delta = 0x4 (option# 3+4=7)
    Option Length = 0x4
    Option Value = 9085
  Option (Uri-Path)
    Option Delta = 0x4 (option# 7+4=11)
    Option Length = 0x5
    Option Value = "est"
  Option (Uri-Path)
    Option Delta = 0x0 (option# 11+0=11)
    Option Length = 0x6
    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 are optional. They are usually omitted as the DTLS destination and port are sufficient. 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. Alternatively, if a UDP port to a server is blocked, someone could send the DTLS packets to a known open port on the server and use the Uri-Port to convey the intended port he is attempting to reach.

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 =
308207b06092a864886f70d010702a082026c308202680201013100300b
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 =
df9c9924b30a06082a8648ce3d0403023067310b300906035504061302
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 = 0x2
    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
e5c7306163b3c3834bab3c103f74307594c089aaa0ac870cd13b902caal
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+4)) which gives SZX=4. The notation for block numbering is the same as in Appendix B.1. It is assumed that CSR takes N1+1 blocks and the cert response takes N2+1 blocks. The header fields and the payload are omitted for brevity.
POST [2001:db8::2:1]:61616/est/sen (CON)(1:0/1/256) {CSR req} -->
   <-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR req} -->
   <-- (ACK) (1:1/1/256) (2.31 Continue)
   .
   .
POST [2001:db8::2:1]:61616/est/sen (CON)(1:N1/0/256) {CSR req} -->
   <-- (ACK) (1:N1/0/256) (2:0/1/256) (2.04 Changed) {Cert resp}
POST [2001:db8::2:1]:61616/est/sen (CON)(2:1/0/256) -->
   <-- (ACK) (2:1/1/256) (2.04 Changed) {Cert resp}
   .
   .
POST [2001:db8::2:1]:61616/est/sen (CON)(2:N2/0/256) -->
   <-- (ACK) (2:N2/0/256) (2.04 Changed) {Cert resp}

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

Breakdown of cacerts response containing one root CA certificate.
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    91:89:bc:df:9c:99:24:4b
Signature Algorithm: ecdsa-with-SHA256
  Issuer: C=US, ST=CA, L=LA, O=Example Inc, OU=certification, CN=Root CA
Validity
  Not Before: Jan 7 10:40:41 2019 GMT
  Not After : Jan 2 10:40:41 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:
      e6:98:97:33:35:20:00:23:dd:ff:78:67:7a:44:3f:
      fd:bb:ef:04:c9
    ASN1 OID: prime256v1
    NIST CURVE: P-256
X509v3 extensions:
  X509v3 Subject Key Identifier:
  X509v3 Authority Key Identifier:
    keyid:
  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
C.2. enroll / reenroll

The breakdown of the 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:
        9f:dc:49:4f:02:8b:cc:80:c0:3f:15:0b:f5:
        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:

The CSR contained a ChallengePassword which is used for POP linking (Section 6).

The breakdown of the issued certificate response 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)
           9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:
           56:38:e5:9f:d9
    ASN1 OID: prime256v1
    NIST CURVE: P-256
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
    X509v3 Subject Key Identifier:
    X509v3 Authority Key Identifier:
      keyid:
    X509v3 Key Usage: critical
      Digital Signature, Key Encipherment
    X509v3 Subject Alternative Name:
      othername:<unsupported>
    Signature Algorithm: ecdsa-with-SHA256

C.3. serverkeygen

  The following is the breakdown of the request example used.
Certificate Request:

Data:

Version: 0 (0x0)
Subject: O=skg example
Subject Public Key Info:
  Public Key Algorithm: id-ecPublicKey
  Public-Key: (256 bit)
    pub:
      04:1b:b8:c1:11:78:96:f9:8e:45:06:c0:3d:70:ef:
      be:82:0d:8e:38:ea:97:e9:d6:5d:52:c8:46:0c:58:
      1a:28:4c:c9:9f
    ASN1 OID: prime256v1
    NIST CURVE: P-256

Attributes:
a0:00

Signature Algorithm: ecdsa-with-SHA256

The following is the breakdown of the private key content of the server-side key generation response payload.

Private-Key: (256 bit)

priv:
  d8:a6
pub:
  04:1b:b8:c1:11:78:96:f9:8e:45:06:c0:3d:70:ef:
  be:82:0d:8e:38:ea:97:e9:d6:5d:52:c8:46:0c:58:
  1a:28:4c:c9:9f

ASN1 OID: prime256v1
NIST CURVE: P-256

The following is the breakdown of the certificate of the second part of the server-side key generation response payload.
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 1327972925857878603 (0x126de8571518524b)
      Signature Algorithm: ecdsa-with-SHA256
      Issuer: O=skg example
      Validity
         Not Before: Jan 9 08:57:08 2019 GMT
         Not After : Jan 4 08:57:08 2039 GMT
      Subject: O=skg example
      Subject Public Key Info:
         Public Key Algorithm: id-ecPublicKey
         Public-Key: (256 bit)
            pub:
               04:1b:b8:c1:11:78:96:f9:8e:45:06:c0:3d:70:ef:
               be:82:0d:8e:38:ea:97:e9:d6:5d:52:c8:46:0c:58:
               1a:28:4c:c9:9f
            ASN1 OID: prime256v1
            NIST CURVE: P-256
      X509v3 extensions:
         X509v3 Basic Constraints:
            CA:FALSE
         Netscape Comment:
            OpenSSL Generated Certificate
         X509v3 Subject Key Identifier:
         X509v3 Authority Key Identifier:
            keyid:
         Signature Algorithm: ecdsa-with-SHA256

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