ACE
Internet-Draft
Intended status: Standards Track
Expires: July 13, 2019

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

January 9, 2019

EST over secure CoAP (EST-coaps)
draft-ietf-ace-coap-est-07

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

Copyright Notice

Table of Contents

1. Change Log

EDNOTE: Remove this section before publication

-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 8. 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 5.7 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 messages run 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 over from [RFC7030]. Consequently, much text is directly traceable to [RFC7030]. The same document structure is followed to point out the differences and commonalities between EST and EST-coaps.

4. Conformance to RFC7925 profiles

This section shows 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] snd discussed in Section 5.7 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] can be used as 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 and thus support for DTLS 1.3 does not mandate point formation 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 allow for the authenticating the server the first time before updating its trust anchor (Explicit TA) [RFC7030].

The authentication of the EST-coaps client is based on 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 reenrollment of clients.

- a previously installed certificate (e.g., manufacturer-installed IDevID (IEEE 802.1AR [ieee802.1ar] certificate or a certificate issued by some other party); the server is expected to trust the previously installed CA certificate in this case. IDevID’s are expected to have a very long life, as long as the device, but under some conditions could expire. In the latter case, the server MAY want to authenticate a client certificate against its trust store although the certificate is expired (Section 11).

Client authentication via DTLS Client Certificate is mandatory.

5. Protocol Design

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to transport CoAP messages in blocks thus avoiding (excessive) fragmentation of UDP datagrams. The use of "Block" for the transfer of larger EST messages is specified in Section 5.5. Figure 1 below shows the layered EST-coaps architecture.

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

Figure 1: EST-coaps protocol layers

The EST-coaps protocol design follows closely the EST design. The actions supported by EST-coaps are identified by their message types:

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

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

- Certificate Signing Request (CSR) Attributes request messages, 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 for an external entity to generate its private key.

5.1. 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 1 specifies the mandatory-to-implement or optional implementation of the est-coaps functions.

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

Table 1: Table 1: List of EST-coaps fuctions

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

5.2. Payload format

The content-format (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 5.4.

EST-coaps is designed for use between 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 DER format. Section 5.2.1 specifies the payload structure when multiple media types are present in the payload.

5.2.1. Content Format application/multipart-core

A representation with content format ID 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 ID of the consecutive representation. For example, a collection containing two representations in response to a server-side key generation request, could include a private key in PKCS#8 [RFC5958] with content format ID 284 (0x011C) and a certificate with content format ID 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

The PKCS#8 key and X.509 certificate representations are ASN.1 encoded in binary DER format. An example is shown in Appendix A.4.

In cases where the private key is further encrypted with CMS (as explained in Section 5.7) the content format ID is 280 (0x0118).

5.3. Message Bindings

The general EST CoAP 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, and Location-Path. These CoAP Options are used to communicate the HTTP fields specified in the EST REST messages.

- EST URLs are HTTPS based (https://), in CoAP these are assumed to be translated to coaps (coaps://)

Appendix A includes some practical examples of EST messages translated to CoAP.

5.4. 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 HTTP POST EST requests (/simpleenroll, /simplereenroll, /serverkeygen). Response code HTTP 202 Retry-After that existed in EST has no equivalent in CoAP. Section 5.6 specifies how EST requests over CoAP handle delayed messages.

EST makes use of HTTP 204 and 404 responses when a resource is not available for the client. The equivalent CoAP error code 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 required COAP option (i.e Content-Format) is omitted, the server is expected to return a 4.02.

5.5. 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 OID fields used. For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, SANs and cert fields. For 384-bit curves, ECDSA certs increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacerts response from the server can include multiple certs 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 certs, 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], blockwise transfers SHOULD be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks. [RFC7959] defines SZX in the block option fields. SZX is used to convey the size of the blocks in the requests or responses. The CoAP client MAY specify the Block1 size and MAY also specify the Block2 size. The CoAP server MAY specify the Block2 size, but not the Block1 size.

[RFC7959] also defines Size1 and Size2 options to provide size information about the resource representation in a request and response. The Size1 response MAY be parsed by the client as a size indication of the Block2 resource in the server response or by the server as a request for a size estimate by the client. Similarly, the Size2 option defined in BLOCK should be parsed by the server as an indication of the size of the resource carried in Block1 options and by the client as a maximum size expected in the 4.13 (Request Entity Too Large) response to a request.

Examples of fragmented EST messages are shown in Appendix B.

5.6. Delayed Responses

Server responses can sometimes be delayed. According to section 5.2.2 of [RFC7252], a slow server can acknowledge the request with a 2.31 code and respond later with the requested resource representation. In particular, a slow server can respond to an 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 where a client sends an enrollment request that uses more than one "Block1" blocks. The server uses an empty 0.00 ACK to announce the delayed response which is provided later with 2.04 messages containing "Block2" options. Having received the first 256 bytes in the first "block2" block, the client asks for a block reduction to 128 bytes in all following "block2" blocks, starting with the second block (NUM=1).

```
   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} -->
          <-- (0.00 empty ACK)
                         |
          ...... short delay before certificate is ready ......
                         |
         <-- (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 enrolment 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 can resend the enrolment 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 enrolment request that uses more than one "Block1" blocks to send the CSR to the server. The server needs more than one "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. Note that in the example below 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) (2.04 Changed){Cert resp}
   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 3: EST-COAP enrolment with long wait

5.7. 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 is 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.

[RFC7030] recommends the private key returned by the server to be encrypted. This specification provides two methods to encrypt the generated key, symmetric and asymmetric. The methods are signalled 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 this specification.

The sever-side key generation response is returned using a CBOR array Section 5.2.1. The certificate part exactly matches the response from an enrollment response. The private key can be in unprotected PKCS#8 [RFC5958] format (content type 281) or protected inside of CMS SignedData (content type 280). The SignedData is signed by the party that generated the private key, which may or may not 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 symmetricly 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 using the client defined (in the CSR) asymmetric public key and is carried in an encryptedKey attribute in a KeyTransRecipientInfo. 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 using 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 type 281). Symmetric or asymmetric encryption of the private key (CMS EnvelopedData, content type 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).

5.8. Deployment limits

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.

6. 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]. The 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 2. The ArbitraryLabel Path-Segment, if used, SHOULD be of the shortest length possible (Sections 3.1 and 3.2.2 of [RFC7030]. Following [RFC7030] discovery is not needed when the client is preconfigured with the /.well-known/est server URI and the coaps port 5684.

The EST-coaps server URIs, obtained through discovery of the EST-coaps root 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 2 provides the mapping from the EST URI path to the shorter EST-coaps URI path.

EST	EST-coaps
/cacerts	/crts
/simpleenroll	/sen
/simplereenroll	/sren
/csrattrs	/att
/serverkeygen	/skg

Table 2: Table 2: 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 server MAY return all available resource paths and the used content types. This is useful when multiple content types are supported by the EST-coaps server and optional functions are available. 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
</est>; rt="ace.est",
</est/crts>;rt="ace.est.crts";ct=281,
</est/sen>;rt="ace.est.sen";ct=281 286,
</est/sren>;rt="ace.est.sren";ct=281 286,
</est/att>;rt="ace.est.att";ct=285,
</est/skg>;rt="ace.est.skg";ct=280 286 62
```

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 from multiple content types.

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.

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

7. 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. Where TLS is used in the context of EST, it is understood that EST-coaps uses DTLS instead. No other changes are necessary regarding the secure transport of EST messages (all provisional modes etc. are the same as in TLS).

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 descrbed 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 is supposed to add this "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request [RFC5967] Info 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 [RFC6347]. The Finished message is calculated as:

```
PRF(master_secret, finished_label, Hash(handshake_messages))
   [0..verify_data_length-1];
```

Similarly, for DTLS 1.3, the Finished message MUST be computed as if each handshake message had been sent as a single fragment following the algorithm described in 4.4.4 of [RFC8446]. The Finished message is calculated as:

```
HMAC(finished_key,
    Transcript-Hash(Handshake Context,
    Certificate*, CertificateVerify*))

    * Only included if present.
```

In a constrained CoAP environment, endpoints can’t 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. The DTLS connection SHOULD remain open for sequential EST transactions. For example, an EST cacerts request that is followed by a simpleenroll request can use the same authenticated DTLS connection. However, some additional security considerations apply regarding the use of the Implicit and Explicit TA database (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.rescorla-tls-dtls-connection-id] negotiates a connection ID that can eliminate the need for new handshake and its additional cost.

8. 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 that supports TLS/HTTP. 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.

```
                                        Constrained Network
   .------.                         .----------------------------.
   |  CA  |                         |.--------------------------.|
   '------'                         ||                          ||
      |                             ||                          ||
   .------.  HTTP   .------------------.   CoAPS  .-----------.  ||
   | 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 needs to be 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 7). 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 2 contains the URI mappings between EST-coaps and EST that the Registrar MUST adhere to. Section 5.4 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-Type 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 should be included in 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 6. The available actions of the Registrars MUST be announced with as many resource paths necessary.
.

9. Parameters

This section addresses transmission parameters described in sections 4.7 and 4.8 of [RFC7252].

ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 byte/second

EST does not impose any unique parameters that affect the CoAP parameters But the CoAP ones could be affecting EST. For example, the processing delay of CAs could be less then 2s, but in this case the EST-coaps server should be sending a CoAP ACK every 2s while processing.

The main recommendation, based on experiments, is to follow the default CoAP configuration parameters. However, depending on the implementation scenario, retransmissions and timeouts can also occur on other networking layers, governed by other configuration parameters.

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.

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 temporarily in the Expert Review range (0-255).

HTTP Media-Type	ID	Reference
application/pkcs7-mime; smime-type=server-generated-key	280	[I-D.ietf-lamps-rfc5751-bis] [RFC7030]
application/pkcs7-mime; smime-type=certs-only	281	[I-D.ietf-lamps-rfc5751-bis]
application/pkcs7-mime; smime-type=CMC-request	282	[I-D.ietf-lamps-rfc5751-bis] [RFC5273]
application/pkcs7-mime; smime-type=CMC-response	283	[I-D.ietf-lamps-rfc5751-bis] [RFC5273]
application/pkcs8	284	[I-D.ietf-lamps-rfc5751-bis] [RFC5958]
application/csrattrs	285	[RFC7030] [RFC7231]
application/pkcs10	286	[I-D.ietf-lamps-rfc5751-bis] [RFC5967]

Table 3: New CoAP Content-Formats

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.
- rt="ace.est.crts". This resource depicts the support of EST get cacerts.
- rt="ace.est.sen". This resource depicts the support of EST simple enroll.
- rt="ace.est.sren". This resource depicts the support of EST simple reenroll.
- rt="ace.est.att". This resource depicts the support of EST CSR attributes.
- 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 be carefully managed to reduce the chance of a third-party CA with poor certification practices from being trusted. 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 persistent DTLS connection where a /sen request follows a /crt in the same connection, a client MAY choose to keep the connection already authenticated by the Implicit TA open for efficiency reasons (Section 7) by assuming that the identity of the server is to be trusted. In that case then the Explicit TA MUST be used starting from the next DTLS connection.

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.

Regarding the Certificate Signing Request (CSR), an adversary could exclude attributes that a server may want, include attributes that a server may not want, and render meaningless other attributes that a server may want. The 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 8 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 7) 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 5.7 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]
Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for CoAP", draft-ietf-core-multipart-ct-02 (work in progress), August 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-dtls13-30 (work in progress), November 2018.

[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>.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[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>.

14.2. Informative References

[COREparams] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml>.

[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.moskowitz-ecdsa-pki]
Moskowitz, R., Birkholz, H., Xia, L., and M. Richardson, "Guide for building an ECC pki", draft-moskowitz-ecdsa-pki-04 (work in progress), September 2018.

[I-D.rescorla-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., Fossati, T., and T. Gondrom, "The Datagram Transport Layer Security (DTLS) Connection Identifier", draft-rescorla-tls-dtls-connection-id-02 (work in progress), November 2017.

[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] Nadia Heninger, Zakir Durumeric, Eric Wustrow, J. Alex Halderman, "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>.

[RFC5273] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC): Transport Protocols", RFC 5273, DOI 10.17487/RFC5273, June 2008, <https://www.rfc-editor.org/info/rfc5273>.

[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>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/info/rfc5958>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <https://www.rfc-editor.org/info/rfc6090>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[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>.

[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>.

[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>.

[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>.

[RSAorig] Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, Vashek Matyas, "The Million-Key Question - Investigating the Origins of RSA Public Keys", USENIX Security Symposium 2016 ISBN 978-1-931971-32-4, August 2016.

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 DER 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 DER formatted and transferred over an encrypted DTLS tunnel. The hexadecimal representations in the examples below would NOT be transported in hex, but in binary DER. Hex is used for visualization purposes because a binary representation cannot be rendered well in text.

The message content breakdown is presented in Appendix C.

The corresponding CoAP headers are only shown in Appendix A.1. Creating CoAP headers is assumed to be generally understood.

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

A.1. cacerts

In EST-coaps, a coaps cacerts message can be:

```
GET coaps://192.0.2.1:8085/est/crts
```

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           [optional]
        Option Delta = 0x3  (option# 3 Uri-Host)
        Option Length = 0x9
        Option Value = 192.0.2.1
      Option           [optional]
        Option Delta = 0x4  (option# 3+4=7 Uri-Port)
        Option Length = 0x4
        Option Value = 8085
      Option
        Option Delta = 0x4   (option# 7+4=11 Uri-Path)
        Option Length = 0x5
        Option Value = "est"
      Option
        Option Delta = 0x0   (option# 11+0=11 Uri-Path)
        Option Length = 0x6
        Option Value = "crts"
      Option
        Option Delta = 0x3   (option# 11+3=14 Max-Age)
        Option Length = 0x1
        Option Value = 0x1    (1 minute)
     Payload = [Empty]
```

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 DER format}
```

with CoAP fields

```
     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

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

     Payload =
   3082027b06092a864886f70d010702a082026c308202680201013100300b
   06092a864886f70d010701a082024e3082024a308201f0a0030201020209
   009189bcdf9c99244b300a06082a8648ce3d0403023067310b3009060355
   040613025553310b300906035504080c024341310b300906035504070c02
   4c413114301206035504 0a0c0b4578616d706c6520496e63311630140603
   55040b0c0d63657274696669636174696f6e3110300e06035504030c0752
   6f6f74204341301e170d3139303130373130343034315a170d3339303130
   323130343034315a3067310b3009060355040613025553310b3009060355
   04080c024341310b300906035504070c024c41311430120603550 40a0c0b
   4578616d706c6520496e633116301406035504 0b0c0d6365727469666963
   6174696f6e3110300e06035504030c07526f6f742043413059301306072a
   8648ce3d020106082a8648ce3d030107034200048149940 82b6e8185f3df
   53f5e0bee698973335200023ddf78cd17a443ffd8ddd40908769c55652ac
   2ccb75c4a50a7c7ddb7c22dae6c85cca538209fdbbf104c9a38184308181
   301d0603551d0e041604142495e816ef6ffcaaf356ce4adffe33cf492abb
   a8301f0603551d230418301680142495e816ef6ffcaaf356ce4adffe33cf
   492abba8300f0603551d130101ff040530030101ff300e0603551d0f0101
   ff040403020106301e0603551d1104173015811363657274696679406578
   616d706c652e636f6d300a06082a8648ce3d0403020348003045022100da
   e37c96f154c32ec0b4af52d46f3b7ecc9687ddf267bcec368f7b7f135327
   2f022047a28ae5c7306163b3c3834bab3c103f743070594c089aaa0ac870
   cd13b902caa1003100

   The breakdown of the payload is shown in Appendix C.1.
```

A.2. csrattrs

In the following csrattrs exchange, the CoAP GET request looks like

```
REQ:
GET coaps://[2001:db8::2:1]:61616/est/att
(Content-Format: 285)
```

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

```
307c06072b06010101011630220603883701311b13195061727365205345
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 DER format. The EST-coaps server returns attributes that the client can ignore if they are unknown to him.

A.3. enroll / reenroll

During the (re-)enroll exchange the EST-coaps client uses a CSR (Content-Format 286) request in the POST request payload. As shown in Appendix C.2, the CSR contains a ChallengePassword which is used for POP linking (Section 7).

```
POST [2001:db8::2:1]:61616/est/sen
(token 0x45)
(Content-Format: 286)

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

308201853082012c0201003070310b3009060355040613025553310b3009
06035504080c024341310b300906035504070c024c413114301206035504
0a0c0b6578616d706c6520496e63310c300a060355040b0c03496f543112
301006035504030c09436c69656e74205241310f300d0603550405130657
74313233343059301306072a8648ce3d020106082a8648ce3d0301070342
00041bb8c1117896f98e4506c03d70efbe820d8e38ea97e9d65d52c8460c
5852c51dd89a61370a2843760fc859799d78cd33f3c1846e304f1717f812
3f1a284cc99fa05a301b06092a864886f70d010907310e0c0c6461746e69
65746465657274303b06092a864886f70d01090e312e302c302a0603551d
1104233021a01f06082b06010505070804a013301106092b06010401b43b
0a01040401020304300a06082a8648ce3d040302034700304402201f82c6
868a654e2dec43cff50aebd6cbbe20dc8242a20a806684f2b8545d008902
20668de2c306df1768105a781e49b1cdc42a2a7f41d6b71d928789547d61
b2b7cf
```

After verification of the CSR by the server, a 2.01 Content response with the issued certificate will be returned to the client. As described in Section 5.6, 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.

```
RET:
(Content-Format: 281)(token =0x45)
2.01 Created
```

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

```
3082028206092a864886f70d010702a08202733082026f0201013100300b
06092a864886f70d010701a082025530820251308201f7a0030201020209
00ce06119a0fd27ca9300a06082a8648ce3d040302305d310b3009060355
040613025553310b300906035504080c02434131143012060355040a0c0b
4578616d706c6520496e6331163014060355040b0c0d6365727469666963
6174696f6e3113301106035504030c0a3830322e3141522043413020170d
3139303130373130343832345a180f39393939313233313233353935395a
3070310b3009060355040613025553310b300906035504080c024341310b
300906035504070c024c4131143012060355040a0c0b6578616d706c6520
496e63310c300a060355040b0c03496f543112301006035504030c09436c
69656e74205241310f300d06035504051306577431323334305930130607
2a8648ce3d020106082a8648ce3d030107034200041bb8c1117896f98e45
06c03d70efbe820d8e38ea97e9d65d52c8460c5852c51dd89a61370a2843
760fc859799d78cd33f3c1846e304f1717f8123f1a284cc99fa3818a3081
8730090603551d1304023000301d0603551d0e04160414494be598dc8dbc
0dbc071c486b777460e5cce621301f0603551d23041830168014d344161b
ff1fa5343015958577dd33507be6b29b300e0603551d0f0101ff04040302
05a0302a0603551d1104233021a01f06082b06010505070804a013301106
092b06010401b43b0a01040401020304300a06082a8648ce3d0403020348
003045022100a8073d6c1f9abb40739fc85a3773378568544036d8cd24f0
1d4b34cb61d9602c022008cc77f8dd5ca7c2fcf95ffc94fdc341e2b61080
118a9576c09e88d2fbd8a921a1003100
```

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

A.4. serverkeygen

In a serverkeygen exchange the CoAP GET request looks like

```
POST coaps://192.0.2.1:8085/est/skg
(token 0xa5)
(Content-Format: 286)(Max-Age=120)
```

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

```
3081cf3078020100301631143012060355040a0c0b736b67206578616d70
6c653059301306072a8648ce3d020106082a8648ce3d030107034200041b
b8c1117896f98e4506c03d70efbe820d8e38ea97e9d65d52c8460c5852c5
1dd89a61370a2843760fc859799d78cd33f3c1846e304f1717f8123f1a28
4cc99fa000300a06082a8648ce3d04030203470030440220387cd4e9cf62
8d4af77f92ebed4890d9d141dca86cd2757dd14cbd59cdf6961802202f24
5e828c77754378b66660a4977f113cacdaa0cc7bad7d1474a7fd155d090d
```

The response would follow [I-D.ietf-core-multipart-ct] and could looke like

```
   RET:
   2.01 Content (Content-Format: 62)
   (token=0xa5)

   [ The hexadecimal representations below would NOT be transported
   in hex, but in DER. 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
   6b02010104200b9a67785b65e07360b6d28cfc1d3f3925c0755799deeca7
   45372b01697bd8a6a144034200041bb8c1117896f98e4506c03d70efbe82
   0d8e38ea97e9d65d52c8460c5852c51dd89a61370a2843760fc859799d78
   cd33f3c1846e304f1717f8123f1a284cc99f
   19 0119                               # unsigned(281)
   59 01D3                               # bytes(467)
   308201cf06092a864886f70d010702a08201c0308201bc0201013100300b
   06092a864886f70d010701a08201a23082019e30820143a0030201020208
   126de8571518524b300a06082a8648ce3d04030230163114301206035504
   0a0c0b736b206578616d706c65301e170d3139303130393038353730385a
   170d3339303130343038353730385a301631143012060355040a0c0b73
   6b206578616d706c653059301306072a8648ce3d020106082a8648ce3d
   030107034200041bb8c1117896f98e4506c03d70efbe820d8e38ea97e9d6
   5d52c8460c5852c51dd89a61370a2843760fc859799d78cd33f3c1846e30
   4f1717f8123f1a284cc99fa37b307930090603551d1304023000302c0609
   6086480186f842010d041f161d4f70656e53534c2047656e657261746564
   204365727469666963617465301d0603551d0e04160414494be598dc8dbc
   0dbc071c486b777460e5cce621301f0603551d23041830168014494be598
   dc8dbc0dbc071c486b777460e5cce621300a06082a8648ce3d0403020349
   003046022100a4b167d0f9add9202810e6bf6a290b8cfdfc9b9c9fea2cc1
   c8fc3a464f79f2c202210081d31ba142751a7b4a34fd1a01fcfb08716b9e
   b53bdaadc9ae60b08f52429c0fa1003100
```

The breakdown of the request and response is shown in Appendix C.3

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 DER 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 with confirmable (CON) option and the content format of the response, even though not shown, is 281 (application/pkcs7-mime; smime-type=certs-only). The transer of the 11 blocks with partially filled block NUM=10 is shown below

```
   GET /192.0.2.1:8085/est/crts   (2:0/0/64)    -->
                 <--   (2:0/1/64) 2.05 Content
   GET /192.0.2.1:8085/est/crts   (2:1/0/64)    -->
                 <--   (2:1/1/64) 2.05 Content
                         |
                         |
                         |
   GET /192.0.2.1:8085/est/crts    (2:10/0/64)  -->
                 <--   (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           [optional]
        Option Delta = 0x3  (option# 3 Uri-Host)
        Option Length = 0x9
        Option Value = 192.0.2.1
      Option           [optional]
        Option Delta = 0x4   (option# 3+4=7 Uri-Port)
        Option Length = 0x4
        Option Value = 8085
      Option
        Option Delta = 0x4    (option# 7+4=11 Uri-Path)
        Option Length = 0x5
        Option Value = "est"
      Option4
        Option Delta = 0x0    (option# 11+0=11 Uri-Path)
        Option Length = 0x6
        Option Value = "crts"
     Payload = [Empty]
```

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 DER. 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 DER. Hex is used because a binary representation
  cannot be rendered well in text. ]

  Payload =
df9c99244b300a06082a8648ce3d0403023067310b300906035504061302
5553310b300906035504080c024341310b300906035504070c024c413114
30120603
```

The 11th 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 DER. Hex is used because a binary representation
  cannot be rendered well in text. ]

  Payload =
2ec0b4af52d46f3b7ecc9687ddf267bcec368f7b7f1353272f022047a28a
e5c7306163b3c3834bab3c103f743070594c089aaa0ac870cd13b902caa1
003100
```

B.2. enroll

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 enrolment 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:
                    04:81:49:94:08:2b:6e:81:85:f3:df:53:f5:e0:be:
                    e6:98:97:33:35:20:00:23:dd:f7:8c:d1:7a:44:3f:
                    fd:8d:dd:40:90:87:69:c5:56:52:ac:2c:cb:75:c4:
                    a5:0a:7c:7d:db:7c:22:da:e6:c8:5c:ca:53:82:09:
                    fd:bb:f1:04:c9
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Key Identifier:
24:95:E8:16:EF:6F:FC:AA:F3:56:CE:4A:DF:FE:33:CF:49:2A:BB:A8
            X509v3 Authority Key Identifier:
                keyid:
24:95:E8:16:EF:6F:FC:AA:F3:56:CE:4A:DF:FE:33:CF:49:2A:BB:A8

            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
         30:45:02:21:00:da:e3:7c:96:f1:54:c3:2e:c0:b4:af:52:d4:
         6f:3b:7e:cc:96:87:dd:f2:67:bc:ec:36:8f:7b:7f:13:53:27:
         2f:02:20:47:a2:8a:e5:c7:30:61:63:b3:c3:83:4b:ab:3c:10:
         3f:74:30:70:59:4c:08:9a:aa:0a:c8:70:cd:13:b9:02:ca
```

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, CN=Client RA/serialNumber=Wt1234
           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:
                       52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
                       9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
                       1a:28:4c:c9:9f
                   ASN1 OID: prime256v1
                   NIST CURVE: P-256
           Attributes:
               challengePassword        :datnietdeert
           Requested Extensions:
               X509v3 Subject Alternative Name:
                   othername:<unsupported>
       Signature Algorithm: ecdsa-with-SHA256
            30:44:02:20:1f:82:c6:86:8a:65:4e:2d:ec:43:cf:f5:0a:eb:
            d6:cb:be:20:dc:82:42:a2:0a:80:66:84:f2:b8:54:5d:00:89:
            02:20:66:8d:e2:c3:06:df:17:68:10:5a:78:1e:49:b1:cd:c4:
            2a:2a:7f:41:d6:b7:1d:92:87:89:54:7d:61:b2:b7:cf
```

The CSR contained a ChallengePassword which is used for POP linking (Section 7)

The breakdown of the issued certificate response is

```
   Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number:
               ce:06:11:9a:0f:d2:7c:a9
       Signature Algorithm: ecdsa-with-SHA256
           Issuer: C=US, ST=CA, O=Example Inc,
                   OU=certification, CN=802.1AR CA
           Validity
               Not Before: Jan  7 10:48:24 2019 GMT
               Not After : Dec 31 23:59:59 9999 GMT
           Subject: C=US, ST=CA, L=LA, O=example Inc,
                    OU=IoT, CN=Client RA/serialNumber=Wt1234
           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:
                       52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
                       9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
                       1a:28:4c:c9:9f
                   ASN1 OID: prime256v1
                   NIST CURVE: P-256
           X509v3 extensions:
               X509v3 Basic Constraints:
                   CA:FALSE
               X509v3 Subject Key Identifier:
   49:4B:E5:98:DC:8D:BC:0D:BC:07:1C:48:6B:77:74:60:E5:CC:E6:21
               X509v3 Authority Key Identifier:
                   keyid:
   D3:44:16:1B:FF:1F:A5:34:30:15:95:85:77:DD:33:50:7B:E6:B2:9B

               X509v3 Key Usage: critical
                   Digital Signature, Key Encipherment
               X509v3 Subject Alternative Name:
                   othername:<unsupported>
       Signature Algorithm: ecdsa-with-SHA256
            30:45:02:21:00:a8:07:3d:6c:1f:9a:bb:40:73:9f:c8:5a:37:
            73:37:85:68:54:40:36:d8:cd:24:f0:1d:4b:34:cb:61:d9:60:
            2c:02:20:08:cc:77:f8:dd:5c:a7:c2:fc:f9:5f:fc:94:fd:c3:
            41:e2:b6:10:80:11:8a:95:76:c0:9e:88:d2:fb:d8:a9:21
```

C.3. serverkeygen

The followng 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:
                    52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
                    9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
                    1a:28:4c:c9:9f
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            a0:00
    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:38:7c:d4:e9:cf:62:8d:4a:f7:7f:92:eb:ed:48:
         90:d9:d1:41:dc:a8:6c:d2:75:7d:d1:4c:bd:59:cd:f6:96:18:
         02:20:2f:24:5e:82:8c:77:75:43:78:b6:66:60:a4:97:7f:11:
         3c:ac:da:a0:cc:7b:ad:7d:14:74:a7:fd:15:5d:09:0d
```

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

```
Private-Key: (256 bit)
priv:
    0b:9a:67:78:5b:65:e0:73:60:b6:d2:8c:fc:1d:3f:
    39:25:c0:75:57:99:de:ec:a7:45:37:2b:01:69:7b:
    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:
    52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
    9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
    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:
                       52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
                       9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
                       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:
   49:4B:E5:98:DC:8D:BC:0D:BC:07:1C:48:6B:77:74:60:E5:CC:E6:21
               X509v3 Authority Key Identifier:
                   keyid:
   49:4B:E5:98:DC:8D:BC:0D:BC:07:1C:48:6B:77:74:60:E5:CC:E6:21

       Signature Algorithm: ecdsa-with-SHA256
            30:46:02:21:00:a4:b1:67:d0:f9:ad:d9:20:28:10:e6:bf:6a:
            29:0b:8c:fd:fc:9b:9c:9f:ea:2c:c1:c8:fc:3a:46:4f:79:f2:
            c2:02:21:00:81:d3:1b:a1:42:75:1a:7b:4a:34:fd:1a:01:fc:
            fb:08:71:6b:9e:b5:3b:da:ad:c9:ae:60:b0:8f:52:42:9c:0f
```

The private key in the response above is without CMS EnvelopedData and has no additional encryption beyond DTLS (Section 5.7).

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