ACE
Internet-Draft
Intended status: Standards Track
Expires: December 23, 2018

P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
S. Kumar
Philips Lighting Research
M. Richardson
SSW
M. Furuhed
Nexus Group
S. Raza
RISE SICS

June 21, 2018

EST over secure CoAP (EST-coaps)
draft-ietf-ace-coap-est-02

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 low-resource constrained devices to use existing EST functionality for provisioning certificates.

Status of This Memo

Table of Contents

1. 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 [RFC5246], HTTP [RFC7230] and TCP.

EST messages may be relatively large and for this reason this document also uses CoAP Block-Wise Transfer [RFC7959] to offer a fragmentation mechanism of EST messages at the CoAP layer. COAP Observe options [RFC7641] are also used to convey delayed EST responses to clients.

This specification 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.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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.

3. 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 request for a trusted certificate list. Private keys can be transported as responses to a request to a server-side keygeneration as described in section 4.4 of [RFC7030] and discussed in Section 4.5 of this document.

As per [RFC7925] section 3.3 and section 4.4, the mandatory cipher suite for DTLS in EST-coaps is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 defined in [RFC7251], and the the curve secp256r1 MUST be supported [RFC4492]; 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 applies.

DTLS1.2 implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC4492]. Uncompressed point format MUST also be supported. [RFC6090] can be used as summary of the ECC algorithms. DTLS 1.3 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 EST-coaps client MUST be configured with at least an implicit TA database from its manufacturer. The authentication of the EST-coaps server by the EST-coaps client is based on certificate authentication in the DTLS handshake.

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 re-enrollment of clients;

- a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party); the server is expected to trust the manufacturer’s root CA certificate in this case.

4. 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 4.6. The 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 actions supported by EST-coaps are identified by their message types:

- o CA certificate retrieval, needed to receive the complete set of CA certificates.
- o Simple enroll and reenroll, for CA to sign public client-identity key.
- o Certificate Signing Request (CSR) Attributes request messages, informs the client of the fields to include in generated CSR.
- o 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.

4.1. 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 used for CoAP MUST map to an allowed combination of URI and media type as defined for EST. The required content-formats for these requests and response messages are defined in Section 9. The CoAP response codes are defined in Section 4.3.

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. Therefore, the content formats specification in Section 4.1.1 specifies that the binary payload is transported as a CBOR major type 2, a byte string, for all EST-coaps Content-Formats. In the examples of Appendix A, the base16 diagnostic notation is used for CBOR major type 2, where h’450aafbb’ represents an example binary payload.

4.1.1. Content Format application/multipart-core

A representation with content format ID TBD8 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.fossati-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 following representation.

For example, a collection containing two representations, one with content format ID TBD5 and one with content format ID TBD2, looks like this in diagnostic CBOR notation: [TBD5,h’0123456789abcdef’,TBD2,h’fedcba9876543210’]. An example is shown in Appendix A.4.

4.2. Message Bindings

The general EST CoAP message characteristics are:

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

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

- o The CoAP options used are Uri-Host, Uri-Path, Uri-Port, Content-Format, and Location-Path in CoAP. These CoAP Options are used to communicate the HTTP fields specified in the EST REST messages.

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

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

4.3. CoAP response codes

Section 5.9 of [RFC7252] specifies 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, 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 POST EST requests. Response code HTTP 202 has no equivalent in CoAP. In Section 4.4 it is specified how EST requests over CoAP handle delayed messages.

All other HTTP 2xx response codes are not used by EST. For the following HTTP 4xx error codes that may occur: 400, 401, 403, 404, 405, 406, 412, 413, 415; the equivalent CoAP response code for EST-coaps is 4.xx. For the HTTP 5xx error codes: 500, 501, 502, 503, 504 the equivalent CoAP response code is 5.xx.

4.4. Delayed Results

If the server is slow providing the response, she can respond with an empty ACK, sending the content later, according to [RFC7252], section 5.2.2. If the response will be more than one packet (requring block mode) then the client needs to send an empty ACK with code 0.00 for the first block and acknowledge the rest of the blocks accordingly. To demonstrate this situation below we show a client sending an enrollment request that will use more than one Block1 blocks to send the CSR to the server. The server on the other hand will need more than one Block2 blocks to respond, but will need take some time to provide the response. Thus the server will use a 0.00 ACK for the response which will be provided when ready by using 2.04 messages and Block2 options. Readers should note that the client asks for a decrease in the block size when acknowledging the first Block2.

```
CON | POST 1:0/1/256 (enroll request with CSR) -->
       <--   ACK | 2.31 1:0/1/256
CON | POST 1:1/1/256 (enroll request with CSR)
       <--   ACK | 2.31 1:1/1/256
CON | POST 1:2/0/256 (enroll request with CSR)
       <--   ACK (code 0.00, no payload,
                     to signal delay in the response.
                                 When ready, the server transfers
                                 the response in Block2 blocks.)
CON | 2.04 1:2/0/256 & 2:0/1/128  (Certificate) -->
       <--   ACK (code 0.00, no payload)
       <--   CON | POST 2:1/0/128
ACK | 2.04 2:1/1/128  (Certificate) -->
       <--   CON | POST 2:2/0/128
ACK | 2.04 2:2/0/128  (Certificate) -->
```

[EDNOTE: To update this. HTTP 202 Retry-After in EST needs an equivalent mechanism in EST-coaps. Observe seems like a candidate but after the HTTP 202 the client needs to do a new POST, not a GET, so Observe is not the best option. We could use 2.04 or a new 2.0x with Max-Age to convey the EST Retry-After.] It is possible that responses are not always directly available by the server, and may even require manual intervention to generate the certificate for the server response. Delays of minutes to hours are possible. EST requires the use of an HTTP 202 message with a Retry-After header by the server which signals to the client to attempt the request in a certain amount of time. In EST, each GET request MUST be accompanied by the observe option. When the result is directly available, the client receives the result and forgets about the observe as specified in section 3.6 of [RFC7641]. When a POST response is delayed, the POST returns a 2.01 (Created) response code, having put a value in the Location-Path option. After reception of 2.01 the client does a GET request with the observe option to the newly returned location. Once the delayed result is notified by the server, the client forgets about the observe.

Next to the observe option the server MUST specify the Max-Age option that indicates the maximum waiting time in minutes.

4.5. 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 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. Studies have shown that the same keys are generated by the same model devices deployed on-line.

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.

[RFC7030] recommends for the private key returned by the server to be encrypted. The 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. In the symmetric key case, the key can be established out-of-band or alternatively derived by the established TLS connection as described in [RFC5705].

The sever-side key generation response is returned using a CBOR array Section 4.1.1. The certificate part exactly matches the response from a enrollment response. The private key is placed inside of a CMS SignedData. 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].

4.6. Message fragmentation

DTLS defines fragmentation only for the handshake part 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 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. 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. From [RFC0791] follows that the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Thus, even with ECC certs, EST-coaps messages can still exceed sizes in MTU of 1280 for IPv6 or 60-80 bytes for 6LoWPAN [RFC4919] as explained in section 2 of [RFC7959]. EST-coaps needs to be able to fragment EST messages into multiple DTLS datagrams. Fine-grained fragmentation of EST messages is essential.

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.

The BLOCK draft defines SZX in the Block1 and Block2 option fields. These are 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. As explained in Section 1 of [RFC7959]), blockwise transfers SHOULD be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks.

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, 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 messages are shown in Appendix B.

4.7. Deployment limits

Although EST-coaps paves the way for the utilization of EST for constrained devices on constrained networks, some devices 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/CoAP. It is up to the network designer to decide which devices execute the EST protocol and which do not.

5. Discovery and URI

EST-coaps is targeted to low-resource networks with small packets. Saving header space is important and an additional EST-coaps URI is specified that is shorter than the EST URI.

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

The individual EST-coaps server URIs differ from the EST URI by replacing the scheme https by coaps and by specifying shorter resource path names:

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

The ArbitraryLabel Path-Segment SHOULD be of the shortest length possible.

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

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

Table 1

The short resource URIs MUST be supported. The corresponding longer URIs specified in [RFC7030] MAY be supported.

When discovering the root path for the EST resources, the server MAY return all available resource paths and the used content types. This is useful when multiple content types are specified for EST-coaps server. The example below shows the discovery of the presence and location of management data.

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

  RES: 2.05 Content
</est>; rt="ace.est"
</est/crts>;ct=TBD2
</est/sen>;ct=TBD2 TBD7
</est/sren>;ct=TBD2 TBD7
</est/att>;ct=TBD6
</est/skg>;ct=TBD1 TBD7 TBD8
```

The first line of the discovery response MUST be returned. The five consecutive lines MAY be returned. The return of the content-types in the last four lines allows the client to choose the most appropriate one from multiple content types.

6. DTLS Transport Protocol

EST-coaps depends on a secure transport mechanism over UDP that can secure (confidentiality, authenticity) the exchanged CoAP messages.

DTLS is one such secure protocol. When "TLS" is referred to in the context of EST, it is understood that in EST-coaps, security is provided using DTLS instead. No other changes are necessary (all provisional modes etc. are the same as for 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].

CoAP and DTLS can provide proof of identity for EST-coaps clients and server with simple PKI messages conformant to section 3.1 of [RFC5272]. EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in section 3 of [RFC7030].

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 translates to the contents of the first "Finished" message in the (D)TLS handshake between server and client [RFC5929]. The client is then supposed to add this "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request Info to prove that the client is indeed in control of the private key at the time of the TLS session when performing a /simpleenroll, for example. 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

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

MUST be computed as if each handshake message had been sent as a single fragment [RFC6347]. Similarly, for DTLS 1.3, the Finished message

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

     * Only included if present.
```

MUST be computed as if each handshake message had been sent as a single fragment following the algorithm described in 4.4.4 of [I-D.ietf-tls-tls13].

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 persistent EST connections. For example, an EST cacerts request that is followed by a simpleenroll request can use the same authenticated DTLS connection. 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, such as 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.

Support for Observe CoAP options [RFC7641] is compulsory. The necessity of Observer for long delays (minutes - hours)is explained in Section 4.4. Observe options could also be used by the server to notify clients about a change in the cacerts or csr attributes (resources) and might be an area of future work.

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 a non-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. The EST coaps-to-HTTPS Registrar MUST terminate EST-coaps and authenticate the client downstream and initiate EST connections over TLS upstream.

The Registrar SHOULD authenticate the client downstream and it should be authenticated by the EST server or CA upstream. The Registration Authority (re-)creates the secure connection from DTLS to TLS and vice versa. A trust relationship SHOULD be pre-established between the Registrar and the EST servers to be able to proxy these connections on behalf of various clients.

When enforcing Proof-of-Possession (POP), the (D)TLS 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 can be used by an end-entity or client. In other words, the CSR the client is using needs to include information from the DTLS connection the client establishes with the server. In EST, that information is the

(D)TLS tls-unique value of the (D)TLS session. In the presence of ESTcoaps-to-HTTPS Registrar, the EST-coaps client MUST be authenticated and authorized by the Registrar and the Registrar MUST be authenticated as an EST Registrar client to the EST server. Thus the POP information is lost between the EST-coaps client and the EST server. The EST server becomes aware of the presence of an EST Registrar from its TLS client certificate that includes id-kp-cmcRA [RFC6402] extended key usage extension. 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.

One possible use-case, shown in one figure below, is expected to be deployed in practice:

```
                                           Constrained Network
                  .----------.     .-----------------------------.
                  |    CA    |     |.---------------------------.|
                  '----------'     ||                           ||
                        |          ||                           ||
.------.  HTTP   .-----------------.  CoAPS   .-----------.  ||
| EST  |<------->|ESTcoaps-to-HTTPS|<-------->| EST Client|  ||
|Server|over TLS |   Registrar     |          '-----------'  ||
'------'         '-----------------'                         ||
                                   ||                           ||
                                   |'---------------------------'|
                                   '-----------------------------'
```

ESTcoaps-to-HTTPS Registrar at the CoAP boundary.

Table 1 contains the URI mapping between the EST-coaps and EST the Registrar SHOULD adhere to. Section 7 of [RFC8075] and Section 4.3 define the mapping between EST-coaps and HTTP response codes, that determines how the Registrar translates CoAP response codes from/to HTTP status codes. The mapping from Content-Type to media type is defined in Section 9. The conversion from CBOR major type 2 to base64 encoding needs to be done in the Registrar. Conversion is possible because a TLS link exists between EST-coaps-to-HTTP Registrar and EST server and a corresponding DTLS link exists between EST-coaps-to-HTTP Registrar and EST client.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP Registrar SHOULD reassemble the BLOCKs before translating the binary content to Base-64, 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 of Section 5. The available actions of the Registrars MUST be announced with as many resource paths. The discovery of EST server in the http environment follow the rules specified in [RFC7030].

When server-side key generation is used, if the private key is protected using symmetric keys then the Registrar needs to encrypt the private key down to the client with one symmetric key and decrypt it from the server with another. If no private key encryption takes place the Registrar will be able to see the key as it establishes a separate connection to the server. In the case of asymmetrically encrypted private key, the Registrar may not be able to decrypt it if the server encrypted it with a public key that corresponds to a private key that belongs to the client.

8. Parameters

THis section addresses transmission parameters described in sections 4.7 and 4.8 of the CoAP document [RFC7252].

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

Figure 2: EST-COAP protocol parameters

EST does not impose any unique parameters that affect the CoAP parameters in Table 2 and 3 in the CoAP draft but the ones in CoAP could be affecting EST. For example, the processing delay of CAs could be less then 2s, but in this case they should send a CoAP ACK every 2s while processing.

The main recommendation, based on experiments using Nexus Certificate Manager with Californium for CoAP support, communicating with a ContikiOS and tinyDTLS based client, from RISE SICS, is to start with the default CoAP configuration parameters.

However, depending on the implementation scenario, resending 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]:

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

- o NSTART: Limit the number of simultaneous outstanding interactions that a client maintains to a given server. The default is one, hence is the risk of congestion or out-of-order messages already limited.

- o 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 the PROBING_RATE setting is not applicable.

Finally, the Table 3 parameters are mainly derived from the more basic Table 2 parameters. If the CoAP implementation allows setting them directly, they might need to be updated if the table 2 parameters are changed.

9. IANA Considerations

9.1. Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are specified in Table 2. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

Media type	Encoding	ID	Reference
application/pkcs7-mime; smime-type=server-generated-key	-	TBD1	[RFC5751] [RFC7030]
application/pkcs7-mime; smime-type=certs-only	-	TBD2	[RFC5751]
application/pkcs7-mime; smime-type=CMC-request	-	TBD3	[RFC5751] [RFC5273]
application/pkcs7-mime; smime-type=CMC-response	-	TBD4	[RFC5751] [RFC5273]
application/pkcs8	-	TBD5	[RFC5751] [RFC5958]
application/csrattrs	-	TBD6	[RFC7030] [RFC7231]
application/pkcs10	-	TBD7	[RFC5751] [RFC5967]
application/multict	-	TBD8	[I-D.fossati-core-multipart-ct]

Table 2: New CoAP Content-Formats

9.2. Resource Type registry

Additions to the sub-registry "CoAP Resource Type", within the "CoRE Parameters" registry are needed for a new resource type.

o rt="ace.est" needs registration with IANA.

10. Security Considerations

10.1. EST server considerations

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

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 to encrypt its certificate. The transport of these keys is inherently risky. A full probability analysis MUST be done to establish whether server side key generation enhances or decreases the probability of identity stealing.

When a client uses the Implicit TA database for certificate validation, the client cannot verify that the implicit database can act as an RA. It is RECOMMENDED that such clients include "Linking Identity and POP Information" Section 6 in requests (to prevent such requests from being forwarded to a real EST server by a man in the middle). It is 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.

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 certification request links the proof-of-possession to the TLS proof-of-identity. This implies but does not prove that 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.

10.2. 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 is used the Registrar terminating the TLS connection establishes a new one with the upstream CA. Thus, it is impossible for POP to be enforced throughout 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.

In a server-side key generation case, depending on the private key encryption method, the Registrar may be able see the private key as it acts as a man-in-the-middle. Thus, the clients puts its trust on the Registrar not exposing the private key.

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 the random number generation using proper entropy. Since the client has no knowledge if the Registrar will be generating the keys and enrolling the certificates with the CA or if the CA will be responsible for generating the keys, the existence of a Registrar requires the client to put its trust on the registrar doing the right thing if it is generating they private keys.

11. 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, and John Manuel.

12. Change Log

-02:

Added paremeter 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 4.5 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

13. References

13.1. Normative References

[I-D.fossati-core-multipart-ct]
Bormann, C., "Multipart Content-Format for CoAP", draft-fossati-core-multipart-ct-05 (work in progress), June 2018.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 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>.

[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>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[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>.

13.2. Informative References

[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.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <https://www.rfc-editor.org/info/rfc4492>.

[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>.

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

[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>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <https://www.rfc-editor.org/info/rfc5929>.

[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>.

[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>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[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>.

Appendix A. EST messages to EST-coaps

This section takes all examples from Appendix A of [RFC7030], changes the payload from Base64 to binary and replaces the http headers by their CoAP equivalents.

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

Binary payload is a CBOR major type 2 (byte array), that is shown with a base16 (hexadecimal) CBOR diagnostic notation.

[EDNOTE: The payloads of the examples need to be re-generated with appropriate tools and example certificates.]

A.1. cacerts

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

In EST-coaps, a coaps cacerts IPv4 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
   Option1 (Uri-Host)               [optional]
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Observe)
     Option Delta = 0x1  (option nr = 3+3=6)
     Option Length = 0x1
     Option Value = 0    (register)
   Option3 (Uri-Port)               [optional]
     Option Delta = 0x1  (option nr = 6+1=7)
     Option Length = 0x4
     Option Value = 8085
   Option4 (Uri-Path)
     Option Delta = 0x4   (option nr = 7+4= 11)
     Option Length = 0x5
     Option Value = "est"
   Option5 (Uri-Path)
     Option Delta = 0x0   (option nr = 11+0= 11)
     Option Length = 0x6
     Option Value = "crts"
   Option6 (Max-Age)
     Option Delta = 0x3   (option nr = 11+3= 14)
     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: TBD2)
   {payload}
```

with CoAP fields

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content)
  Token = 0x9a   (copied by server)
```

```
     Options
      Option1 (Observe)
        Option Delta = 0x6  (option nr =6)
        Option Length = 0x1
        Option Value = 12    ( 12 > 0)
       Option2 (Content-Format)
         Option Delta = 0xC  (option nr = 6+6 =12)
         Option Length = 0x2
         Option Value = TBD2 (defined in this document)

     Payload =
```


h’30233906092a6206734107028c2a30232602010131003000b06092a6206734107018
c0c3020bb302063c20102020900a61e75193b7acc0d06092a620673410105050030
1b311930170603550403131065737445786 16d706c654341204f774f301e170d313
3303530393033353333315a170d3134303530393033353333315a301b3119301706
0355040313106573744578616d706c654341204f774f302062300d06092a6206734
1010105000320 4f0030204a022041003a923a2968bae4aae136ca4e2512c5200680
358482ac39d6f640e4574e654ea35f48b1e054c5da3372872f7a1e429f4edf39584
32efb2106591d3eb783c1034709f251fc86566bda2d541c792389eac4ec9e181f4b
9f596e5ef2679cc321542b11337f90a44df3c85f1516561fa968a1914f265bc0b82
76ebe3106a790d97d34c8c37c74fe1c30b396424664ac426284a9f6022e02693843
6880adfcd95c98ca1dfc2e6d75319b85d0458de28a9d13fb16d620fff7541f6a25d
7daf004355020301000130b040300f0603551d130101f10530030101fc1d0603551
d0e04160414084d321ca0135e77217a486b686b334b00e0603551d0f0101f104030
20106300d06092a6206734101050500032041002 3703b965746a0c2c978666d787a
94f89b495a11f0d369b28936ec2475c0f0855c8e83f823f2b871a1d92282f323c45
904ba008579216cf5223b8b1bc425a0677262047f7700240631c17f3035d1c3780b
2385241cba1f4a6e98e6be6820306b3a786de5a557795d1893822347b5f825d34a7
ad2876f8feba4d525b31066f6505796f71530003431a3e6bbfe788b4565029a7e20
a51107677552586152d051e8eebf383e92288983421d5c5652a4870c3af74b9bdbe
d6b462e2263d30f6d3020c330206bc20102020101300d06092a620673410105050 0
301b3119301706035504031310657374457861 6d706c654341204f774f301e170d3
133303530393033353333325a170d3134303530393033353333325a301b31193017
0603550403131065737445786 16d706c654341204e774f302062300d06092a62067
341010105000320 4f0030204a02204100ef6b677a3247c1fc03d2b9baf113e5e7e1
1f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a7229283a790
8751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b7bd94338
d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562c4f5abb7
b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c768d03b8
076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c14476c37de0f
55033f192a5ad21f9a2a71c20301000134b050300e0603551d0f0101f104030204c
1d0603551d0e04160414112966e304761732fbfe6a2c823c301f0603551d2304183
0165084d321ca0135e77217a486b686b334b00d06092a620673410105050003 2041
00b382ba3355a50e287bae15758b3beff63d34d3e357b90031495d018868e49589b
9faf46a4ad49b1d35b06ef380106677440934663c2cc111c183655f4dc41c0b3401
123d35387389db91f1e1b4131b16c291d35730b3f9b33c7475124851555fe5fc647
e8fd029605367c7e01281bf6617110021b0d10847dce0e9f0ca6c764b6334784055
172c3983d1e3a3a82301a54fcc9b0670c543a1c747164619101ff23b240b2a26394

```
c1f7d38d0e2f4747928ece5c34627a075a8b3122011e9d9158055c28f020c330206
bc20102020102300d06092a6206734101050500301b3119301706035504031310657
3744578616d706c654341204e774e301e170d3133303530393033353333325a170
d3134303530393033353333325a301b311930170603550403131065737445786166d
706c654341204f774e302062300d06092a62067341010105000320 4f0030204a022
041003a923a2968bae4aae136ca4e2512c5200680358482ac39d6f640e4574e654e
a35f48b1e054c5da3372872f7a1e429f4edf3958432efb2106591d3eb783c103470
9f251fc86566bda2d541c792389eac4ec9e181f4b9f596e5ef2679cc321542b1133
7f90a44df3c85f1516561fa968a1914f265bc0b8276ebe3106a790d97d34c8c37c7
4fe1c30b396424664ac426284a9f6022e02693843688 0adfcd95c98ca1dfc2e6d75
319b85d0458de28a9d13fb16d620fff7541f6a25d7daf004355020301000134b050
300e0603551d0f0101f104030204c1d0603551d0e04160414084d321ca0135e7721
7a486b686b334b01f0603551d230418301653112966e304761732fbfe6a2c823c30
0d06092a62067341010505000320410 02e106933a443070acf5594a3a584d08af7e
06c295059370a06639eff9bd418d13bc25a298223164a6cf1856b11a81617282e4a
410d82ef086839c6e2356903227630654553 51e4c596acc7c016b225dec094706c2
a10608f403b10821984c7c152343b18a768c2ad30238dc45dd653ee6092b0d5cd4c
2f7d236043269357f76d13f95fb5f00d0e19263c6833948e1ba612ce8197af650e2
5d882c12f4b6b9b67252c608ef064aca3f9bc867d71172349d510bb7651cd438837
73d927deb41c4673020bb302063c20102020 9009b9dda3324700d06092a62067341
01050500301b3119301706035504031310657374457861 6d706c654341204e774e3
01e170d3133303530393033353333325a170d3134303530393033353333325a301b
311930170603550403131065737445786166d706c654341204e774e302062300d060
92a62067341010105000320 4f0030204a02204100ef6b677a3247c1fc03d2b9baf1
13e5e7e11f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a722
9283a7908751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b
7bd94338d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562
c4f5abb7b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c
768d03b8076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c1447
6c37de0f55033f192a5ad21f9a2a71c20301000130b040300f0603551d130101f10
530030101fc1d0603551d0e04160414112966e304761732fbfe6a2c823c300e0603
551d0f0101f10403020106300d06092a62067341010505000320410 0423f06d4b76
0f4b42744a279035571696f272a0060f1325a40898509601ad14004f652db6312a1
475c4d7cd50f4b269035585d7856c5337765a66b38462d5bdaa7778aab24bbe2815
e37722cd10e7166c50e75ab75a1271324460211991e7445a2960f47351a1a629253
34119794b90e320bc730d6c1bee496e7ac125ce9a1eca595a3a4c54a865e6b623c9
247bfd0a7c19b56077392555c955e233642bec643ae37c166c5e221d797aea3748f
0391c8d692a5cf9bb71f6d0e37984d6fa673a30d0c006343116f58403100’
```


The hexadecimal dump of the CBOR payload looks like:


```
59 09CD                                   # bytes(2509)
   30233906092A6206734107028C2A302326020101310030 0B06092A62067341070
18C0C3020BB302063C20102020900A61E75193B7ACC0D06092A62067341010505
00301B3119301706035504031310657374457861 6D706C654341204F774F301E1
70D3133303530393033353333315A170D3134303530393033353333315A301B31
193017060355040313106573744578616D706C654341204F774F302062300D060
```

92A620673410101050003204F0030204A022041003A923A2968BAE4AAE136CA4E
2512C5200680358482AC39D6F640E4574E654EA35F48B1E054C5DA3372872F7A1
E429F4EDF3958432EFB2106591D3EB783C1034709F251FC86566BDA2D541C7923
89EAC4EC9E181F4B9F596E5EF2679CC321542B11337F90A44DF3C85F1516561FA
968A1914F265BC0B8276EBE3106A790D97D34C8C37C74FE1C30B396424664AC42
6284A9F6022E026938436880ADFCD95C98CA1DFC2E6D75319B85D0458DE28A9D1
3FB16D620FFF7541F6A25D7DAF004355020301000130B040300F0603551D13010
1F10530030101FC1D0603551D0E04160414084D321CA0135E77217A486B686B33
4B00E0603551D0F0101F104030201006300D06092A6206734101050500032041000
23703B965746A0C2C978666D787A94F89B495A11F0D369B28936EC2475C0F0855
C8E83F823F2B871A1D92282F323C45904BA008579216CF5223B8B1BC425A06772
62047F7700240631C17F3035D1C3780B2385241CBA1F4A6E98E6BE6820306B3A7
86DE5A557795D1893822347B5F825D34A7AD2876F8FEBA4D525B31066F6505796
F71530003431A3E6BBFE788B4565029A7E20A5110767755258 6152D051E8EEBF3
83E92288983421D5C5652A4870C3AF74B9BDBED6B462E2263D30F6D3020C33020
6BC20102020101300D06092A6206734101050500301B311930170603550403131
06573744578616D706C654341204F774F301E170D3133303530393033353333332
5A170D3134303530393033353333325A301B31193017060355040313106573744
578616D706C654341204E774F302062300D06092A620673410101050003204F00
30204A02204100EF6B677A3247C1FC03D2B9BAF113E5E7E11F49E0421120E6B83
84160F2BF02630EF544D5FD0D5623B35713C79A7229283A7908751A634AA420A3
E2A4B1F10519D046F02F5A5DD6D760C2A842356E067B7BD94338D1FAA3B3DDD48
13060A207B0A097067007E45B052B60FDBAE4656E11562C4F5ABB7B0CF87A79D2
21F1127313C53371CE1245D63DB45A1203A23340BA08042C768D03B8076A028D3
A51D87D2EF107BBD6F2305CE5E67668724002FB726DF9C14476C37DE0F55033F1
92A5AD21F9A2A71C20301000134B050300E0603551D0F0101F104030204C1D060
3551D0E04160414112966E304761732FBFE6A2C823C301F0603551D2304183016
5084D321CA0135E77217A486B686B334B00D06092A62067341010505000320410
0B382BA3355A50E287BAE15758B3BEFF63D34D3E357B90031495D018868E49589
B9FAF46A4AD49B1D35B06EF38010667744093466 3C2CC111C183655F4DC41C0B3
401123D35387389DB91F1E1B4131B16C291D35730B3F9B33C7475124851555FE5
FC647E8FD029605367C7E01281BF6617110021B0D10847DCE0E9F0CA6C764B633
4784055172C3983D1E3A3A82301A54FCC9B0670C543A1C747164619101FF23B24
0B2A26394C1F7D38D0E2F4747928ECE5C34627A075A8B3122011E9D9158055C28
F020C330206BC20102020102300D06092A6206734101050500301B31193017060
3550403131065737445786 16D706C654341204E774E301E170D31333035303930
33353333325A170D3134303530393033353333325A301B3119301706035504031
3106573744578616D706C654341204F774E302062300D06092A62067341010105
0003204F0030204A022041003A923A2968BAE4AAE136CA4E2512C520068035848
2AC39D6F640E4574E654EA35F48B1E054C5DA3372872F7A1E429F4EDF3958432E
FB2106591D3EB783C1034709F251FC86566BDA2D541C792389EAC4EC9E181F4B9
F596E5EF2679CC321542B11337F90A44DF3C85F1516561FA968A1914F265BC0B8
276EBE3106A790D97D34C8C37C74FE1C30B396424664AC426284A9F6022E02693
8436880ADFCD95C98CA1DFC2E6D75319B85D0458DE28A9D13FB16D620FFF7541F
6A25D7DAF004355020301000134B050300E0603551D0F0101F104030204C1D060
3551D0E04160414084D321CA0135E77217A486B686B334B01F0603551D2304183
01653112966E304761732FBFE6A2C823C300D06092A62067341010505000320410
02E106933A443070ACF5594A3A584D08AF7E06C295059370A06639EFF9BD418D

```
13BC25A298223164A6CF1856B11A81617282E4A410D82EF086839C6E235690322
763065455351E4C596ACC7C016B225DEC094706C2A10608F403B10821984C7C15
2343B18A768C2AD30238DC45DD653EE6092B0D5CD4C2F7D236043269357F76D13
F95FB5F00D0E19263C6833948E1BA612CE8197AF650E25D882C12F4B6B9B67252
C608EF064ACA3F9BC867D71172349D510BB7651CD43883773D927DEB41C467302
0BB302063C201020209009B9DDA3324700D06092A6206734101050500301B3119
30170603550403131065737445786l6D706C654341204E774E301E170D3133303
5303930333353333325A170D313430353039303335333325A301B311930170603
5504031310657374457861 6D706C654341204E774E302062300D06092A6206734
10101050003204F0030204A02204100EF6B677A3247C1FC03D2B9BAF113E5E7E1
1F49E0421120E6B8384160F2BF02630EF544D5FD0D5623B35713C79A7229283A7
908751A634AA420A3E2A4B1F10519D046F02F5A5DD6D760C2A842356E067B7BD9
4338D1FAA3B3DDD4813060A207B0A097067007E45B052B60FDBAE4656E11562C4
F5ABB7B0CF87A79D221F1127313C53371CE1245D63DB45A1203A23340BA08042C
768D03B8076A028D3A51D87D2EF107BBD6F2305CE5E67668724002FB726DF9C14
476C37DE0F55033F192A5AD21F9A2A71C20301000130B040300F0603551D13010
1F10530030101FC1D0603551D0E04160414112966E304761732FBFE6A2C823C30
0E0603551D0F0101F10403020106300D06092A6206734101050500003204100423
F06D4B760F4B42744A279035571696F272A0060F1325A40898509601AD14004F6
52DB6312A1475C4D7CD50F4B269035585D7856C5337765A66B38462D5BDAA7778
AAB24BBE2815E37722CD10E7166C50E75AB75A1271324460211991E7445A2960F
47351A1A62925334119794B90E320BC730D6C1BEE496E7AC125CE9A1ECA595A3A
4C54A865E6B623C9247BFD0A7C19B56077392555C955E233642BEC643AE37C166
C5E221D797AEA3748F0391C8D692A5CF9BB71F6D0E37984D6FA673A30D0C00634
3116F58403100
```


After reception of the 2.05 response, the client can forget the observe.

A.2. csrattrs

In the following valid /csrattrs exchange, the EST-coaps client authenticates itself with a certificate issued by the connected CA.

The initial DTLS handshake is identical to the enrollment example. The IPv6 CoAP GET request looks like:

```
REQ:
GET coaps://[2001:db8::2:1]:61616/est/att
(Content-Format: TBD6)(observe =0)(Max-Age =1)
```

A 2.05 Content response contains attributes which are relevant for the authenticated client. In this example, the EST-coaps server returns two attributes that the client can ignore when they are unknown to him.

A.3. enroll / reenroll

During the Enroll/Reenroll exchange, the EST-coaps client uses a CSR (Content-Format TBD7) request in the POST request payload.

After verification of the CSR by the server, a 2.05 Content response with the issued certificate will be returned to the client. As described in Section 4.4, if the server is not able to provide a response, then it ACKs the GET (with no payload), and the payload will be sent later as part of the OBSERVE processing.

[EDNOTE: When redoing this example, given that proof of possession (POP) is also used, make sure it is obvious that the ChallengePassword attribute in the CSR is valid HMAC output. HMAC-REAL.]

```
POST [2001:db8::2:1]:61616/est/sen
(token 0x45)
(Content-Format: TBD7)(observe 0)
h’30208530206d020100301f311d301b0603550403131464656d6f7374657034203
13336383134313335323020623 00d06092a620673410101050003204f0030204a
022041005d9f4dffd3c5949f646a958436777856095 0b355c35b8e34726dd3764
54231734795b4c09b9c6d75d408311307a81f7adef7f5d241f7d5be85620c5d44
38bbb4242cf215c167f2ccf36c364ea2618a62f0536576369d6304e6a96877224
7d86824f079faac7a6f694cfda5b84c42087dc062d462190c525813f210a036a7
37b4f30d8891f4b75559fb72752453146332d51c937557716ccec624f5125c3a4
447ad3115020048113fef54ad554ee88af09a2583aac9024075113db4990b1786
b871691e0f02030100018701f06092a6206734109073112131 02b72724369722f
372b4559753530543430 0d06092a62067341010505000320410 0441b40177a3a6
5501487735a8ad5d3827a4eaa867013920e2afcda87aa81733c7c0353be47e1bf
a7cda5176e7ccc6be22ae03498588d5f2de3b143f2b1a6175ec544e8e7625af6b
836fd4416894c2e55ea99c6606f69075d6d53475d410729aa6d806afbb9986caf
7b844b5b3e4545f19071865ada007060cad6db26a592d4a7bda7d586b68110962
1707110340755315 5cddc75481e272b5ed553a8593fb7e25100a6f7605085dab4
fc7e0731f0e7fe305703791362d5157e92e6b5c2e3edbcadb40’

RET:
(Content-Format: TBD2)(token =0x45)(observe =12)
2.01 Created
h’3020f806092a62067341070283293020e50201013100300b06092a62067341070
1830b3020c730206fc20102020115300d06092a62067341010505 00301b311930
170603550403131065737445786 16d706c654341204e774e301e170d313330353
03932333135353 35a170d31343035303932333135353 35a301f311d301b060355
0403131464656d6f7374657034203133363831343133353230206230 0d06092a6
2067341010105000320 4f0030204a022041005d9f4dffd3c5949f646a95843677
78560950b355c35b8e34726dd37645423173479 5b4c09b9c6d75d408311307a81
f7adef7f5d241f7d5be85620c5d4438bbb4242cf215c167f2ccf36c364ea2618a
62f0536576369d6304e6a968772247d86824f079faac7a6f694cfda5b84c42087
dc062d462190c525813f210a036a737b4f30d8891f4b75559fb72752453146332
d51c937557716ccec624f5125c3a4447ad3115020048113fef54ad554ee88af09
a2583aac9024075113db4990b1786b871691e0f02030100013 4b050300e060355
1d0f0101f104030204c1d0603551d0e04160414e81d0788aa2710304c5ecd4d1e
065701f0603551d23041830165311 2966e304761732fbfe6a2c823c300d06092a
62067341010505000320410 02910d86f2ffeeb914c046816871de601567d291b4
3fabee0f0e8ff81cea27302a7133e20e9d04029866a8963c7d14e26fbe8a0ab1b
77fbb1214bbcdc906fbc381137ec1de685f79406c3e416b8d82f97174bc691637
5a4e1c4bf744c7572b4b2c6bade9fb35da786392ee0d95e3970542565f3886ad6
7746d1b12484bb02616e63302dc371dc6006e431fb7c457598dd204b367b0b3d3
258760a303f1102db26327f929b7c5a60173e1799491b69150248756026b80553
171e4733ad3d13c0103100’
```

After reception of the 2.01 response the client can forget the observe registration

The same example when delays occur (omitting the payloads in the examples) has a different behavior. The response to the POST is an empty payload with response code 2.01 (Created) that also returns the resource to query. The client issues a GET with an observe and a new token value, and waits for the notification after possibly receiving an empty payload first.

```
POST [2001:db8::2:1]:61616/est/sen
(token 0x45)(observe = 0)
(Content-Format: TBD7)(Max-Age=120)
[payload]
RET:
(token =0x45)(observe =12)(Location-Path=/est/1245)
2.01 Created
[empty payload]

GET [2001:db8::2:1]:61616/est/1245
(token 0x53)
(observe =0)(Max-Age=120)

RET:
(token =0x53)(observe = 5)
2.01 Created
[empty payload]

RET:
(token =0x53)(observe = 6)
(Content-Format: TBD2)
2.04 Changed
[payload]
```

A.4. serverkeygen

During this valid /serverkeygen exchange, the EST-coaps client authenticates itself using the certificate provided by the connected CA.

The initial DTLS handshake is identical to the enrollment example. The CoAP GET request looks like:

[EDNOTE: same comment as HMAC-REAL above applies.]

[EDNOTE: Suggestion to have only one example with complete encrypted payload (the short one) and point out the different fields. Update this example according to the agreed upon solution from Section 4.5.]

```
POST coaps://192.0.2.1:8085/est/skg
(token 0xa5)(observe = 0)
(Content-Format: TBD7)(Max-Age=120)
```

h’3020813020690201003050b313e303c0603550403133573657276657244b6579476
56e2072657120627920636c69656e7420696e2064656d6f207374657020313220
3133363831343139353531193017060355040513105049443a576964676574205
34e3a3130302062300d06092a620673410101050003204f0030204a02204100f4
dfa6c03f7f2766b23776c333d2c0f9d1a7a6ee36d01499bbe6f075d1e38a57e98
ecc197f51b75228454b7f19652332de5e52e4a974c6ae34e1df80b33f15f47d3b
cbf76116bb0e4d3e04a9651218a476a13fc186c2a255e4065ff7c271cff104e47
31fad53c22b21a1e5138bf9ad0187314ac39445949a48805392390e78c7659621
6d3e61327a534f5ea7721d2b1343c7362b37da502717cfc2475653c7a3860c5f4
0612a5db6d33794d755264b6327e3a3263b149628585b85e57e42f6b3277591b0
2030100018701f06092a6206734109073112131064467341586d4a6e6a6f6b427
4447672300d06092a620673410105050003204100472d11007e5a2b2c2023d47a
6d71d046c307701d8ebc9e47272713378390b4ee321462a3dbe54579f5a514f6f
4050af497f428189b63655d03a194ef729f101743e5d03fbc6ae1e84486d1300a
f9288724381909188c851fa9a5059802eb64449f2a3c9e441353d136768da27ff
4f277651d676a6a7e51931b08f56135a2230891fd184960e1313e7a1a9139ed19
28196867079a456cd2266cb754a45151b7b1b939e381be333fea61580fe5d25bf
4823dbd2d6a98445b46305c10637e202856611’

```
RET:
2.01 Content (Content-Format: TBD8)
(observe = 5)(token=0xa5)
```

[TBD5,

h’30213e020100300d06092a6206734101010500042128302124020100022041003
c0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274
dd01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a1
1bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c
0c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d5
45e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704
efec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e31083010
001022041004e6b3f78b7791d6377f33117c17844531c81111fb8000282816264
915565bc7c3f3f643b537a2c69140a31c22550fa97e5132c61b74166b68626704
260620333050f510096b6570f5880e7e1c15dc0ca6ce2b5f187e2325da14ab705
ad004717f3b2f779127b5c535e0cee6a343b502722f2397a26126e0af606b5aa7
f96313511c0b7eb26354f91b82269de62757e3def807a6afdf83ddcbb0614bb7c
542e6975d6456554e7bd9988fbd1930cd44d0e01ee9182ca54539418653150254
1ad1a2a11e5021040bfce554b642c29131e7d65455e83c5406d76771912f758f5
ee3ee36af386f38ffa313c0f661880c5a2b0970485d36f528e7f77a2e55b4ad76
1242d1c2f75939c8061217d31491d305d3e07d6161c43e26f7de4477b1811de92
33dc75b426302104015bf48ac376f52887813461fc54635517bcb67293837053e
8ce1a33da7a35565a75a370dc14555b5316cb55742380350774d769d151ff0456
0214389a232a2258326163167504cfce44cd316f63bb8a52da53a4cb74fd87194
c0844881f791f23b0813ea0921325edd14459d41c8a1593f04316388e40b35fef

```
7d2a195a5930fa54774427ac821eee2c62790d2c17bd192af794c611011506557
83d4efe22185cbd83368786f2b1e68a5a27067e321066f0217b4b6d7971a3c21a
241366b7907187583b51110210336904 7e5cce0b65012200df5ec697b5827575c
db6821ff299d6a69574b31ddf0fbe9245ea2f74396c24b3a7565067e41366423b
5bdd2b2a78194094dbe333f493d159b8e07722f2280d48388db7f1c9f0633bb0e
173de2c3aa1f200af535411c7090210401421e2ea217e37312dcc606f453a6634
f3df4dc31a9e910614406412e70eec9247f10672a500947a64356c015a845a7d1
50e2e3911a2b3b61070a73247166da10bb45474cc97d1ec2bc392524307f35118
f917438f607f18181684376e13a39e07',
TBD2,
h'3020c506092a6206734107028336302 0f20201013100300b06092a62067341070
183183020d430207cc20102020116300d06092a620673410105050 0301b311930
17060355040313106573744578616d706c654341204e774e301e170d313330353
03932333235353 65a170d313430353039323332353536 5a302c312a3028060355
04031321736572766572736964652 06b65792067656e6572617465642 07265737
06f6e7365302062300d06092a6206734101010500032 04f0030204a022041003c
0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274d
d01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a11
bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c0
c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d54
5e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704e
fec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e310830100
0134b050300e0603551d0f0101f104030204c1d0603551d0e04160414764b1bd5
e69935626e476b195a1a8c1f0603551d230418301653112966e304761732fbfe6
a2c823c300d06092a620673410105050003204100474e5100a9cdaaa813b30f48
40340fb17e7d6d6063064a5a7f2162301c464b5a8176623dfb1a4a484e618de1c
3c3c5927cf590f4541233ff3c251e772a9a3f2c5fc6e5ef2fe155e5e385deb846
b36eb4c3c7ef713f2d137ae8be4c022715fd033a818d55250f4e6077718180755
a4fa677130da60818175ca4ab2af1d15563624c51e13dfdcf381881b72327e2f4
9b7467e631a27b5b5c7d542bd2edaf78c0ac294f3972278996bdf673a334ff74c
84aa7d65726310252f6a4f41281ec10ca2243864e3c5743103100']
```

Without the DecryptKeyIdentifier attribute, the response has no additional encryption beyond DTLS.

The response contains first a preamble that can be ignored. The EST-coaps server can use the preamble to include additional explanations, like ownership or support information

Appendix B. EST-coaps Block message examples

Two examples are presented: (1) a cacerts exchange shows the use of Block2 and the block headers, and (2) a enroll exchange shows the Block1 and Block2 size negotiation for request and response payloads.

B.1. cacerts block example

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 valid /cacerts exchange over DTLS. The content length of the cacerts response in appendix A.1 of [RFC7030] is 4246 bytes using base64. This leads to a length of 2509 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 39 packets with a payload of 64 bytes each, followed by a packet of 13 bytes. The client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request 40 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 because used in the response) followed by a colon, and then the block number (NUM), the more bit (M = 0 in lock2 response means 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 is /application/cacerts.

```
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    (2:39/0/64)  -->
              <--   (2:39/0/64) 2.05 Content
```

40 blocks have been sent with partially filled block NUM=39 as last block.

For further detailing the CoAP headers, the first two blocks are written out.

The header of the first GET looks like:

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.1 GET)
  Token = 0x9a    (client generated)
  Options
   Option1 (Uri-Host)            [optional]
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Observe)
     Option Delta = 0x3  (option nr = 3+3=6)
     Option Length = 0x1
     Option Value = 0       (register)
   Option3 (Uri-Port)            [optional]
     Option Delta = 0x4   (option nr = 3+4=7)
     Option Length = 0x4
     Option Value = 8085
   Option4 (Uri-Path)
     Option Delta = 0x4    (option nr = 7+4=11)
     Option Length = 0x5
     Option Value = "est"
   Option5 (Uri-Path)
     Option Delta = 0x0    (option nr = 11+0=11)
     Option Length = 0x6
     Option Value = "crts"
   Option6 (Max-Age)
     Option Delta = 0x3    (option nr = 11+3=14)
     Option Length = 0x1
     Option Value = 0x1    ( 1 minute)
  Payload = [Empty]
```

The header of the first response looks like:

```
     Ver = 1
     T = 2 (ACK)
     Code = 0x45 (2.05 Content)
     Token = 0x9a     (copied by server)
     Options
       Option1 (Observe)
         Option Delta = 0x6  (option nr=6)
         Option Length = 0x1
         Option Value = 12   (12 > 0)
       Option2 (Content-Format)
         Option Delta = 0xC  (option nr =6+6=12)
         Option Length = 0x2
         Option Value = TBD2
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x1
         Option Value = 0x0A (block number = 0, M=1, SZX=2)
     Payload =
   h'30233906092a6206734107028c2a30232602010131003000b06092a6206734107018
   c0c3020bb302063c20102020900a61e75193b7acc0d06092a6206734101'
```

The second Block2:

```
     Ver = 1
     T = 2 (means ACK)
     Code = 0x45 (2.05 Content)
     Token = 0x9a     (copied by server)
     Options
       Option1 (Observe)
         Option Delta = 0x6  (option nr=6)
         Option Length = 0x1
         Option Value = 16   (16 > 12)
       Option2 (Content-Format)
         Option Delta = 0xC  (option nr =6+6=12)
         Option Length = 0x2
         Option Value = TBD2
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x1
         Option Value = 0x1A (block number = 1, M=1, SZX=2)
     Payload =
   h'05050030
   1b3119301706035504031310657374457861 6d706c654341204f774f301e170d313
   3303530393033353333315a170d3134303530393033353333315a'
```

The 40th and final Block2:

```
Ver = 1
T = 2 (means ACK)
Code = 0x45      (2.05 Content)
Token = 0x9a     (copied by server)
Options
  Option1 (Observe)
    Option Delta = 0x6  (option nr=6)
    Option Length = 0x1
    Option Value = 55   (55 > 12)
  Option2 (Content-Format)
    Option Delta = 0xC  (option nr =6+6=12)
    Option Length = 0x2
    Option Value = TBD2
  Option2 (Block2)
    Option Delta = 0xB  (option 23 = 12 + 11)
    Option Length = 0x2
    Option Value = 0x272 (block number = 39, M=0, SZX=2)
Payload = h'73a30d0c006343116f58403100'
```

B.2. enroll block example

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 request/response consists of two parts: part1 containing the CSR transferred to the server, and part2 contains the certificate transferred back to the client. 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 Cert response takes N2+1 blocks. The header fields and the payload are omitted to show the block exchange. The type of payload is shown within curly brackets.

```
   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}
```

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

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Sandeep S. Kumar
Philips Lighting Research
High Tech Campus 7
Eindhoven 5656 AE
NL

Email: ietf@sandeep.de

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/

Martin Furuhed
Nexus Group

Email: martin.furuhed@nexusgroup.com

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se