
Melina Delgado

A Senior Project in Computer Science
Advised by Richard Yang

May 2019
1 Abstract

The Internet of Things (IoT) is what technologists call everyday objects and physical devices that are capable of Internet connection and typically, wireless communication. As IoT devices become more prevalent in our homes and cities, it is important to continue to use public key infrastructure (PKI) to authenticate the electronic transfer of information. One such cryptographic certificate management protocol is Enrollment over Secure Transport (EST) [RFC7030], which describes an X.509 certificate management protocol used for authenticated/authorized endpoint certificate enrollment through a Certificate Authority (CA) or Registration Authority (RA) [1, 2]. EST transports messages over HTTPS secured with Transport Layer Security (TLS). However, EST cannot be supported by low-resource, constrained devices [3]; for example, devices that can be battery powered and unattended for years cannot support EST payloads with HTTP/TLS. EST-coaps is an Internet-Draft protocol to provide certificate provisioning for low-resource devices. The Internet-Draft defines a new transport for EST based on the Constrained Application Protocol (CoAP) over Datagram Transport Layer Security (DTLS) for low-resource devices, instead of HTTP and TLS. The major change from EST to EST-coaps is the reduction in size of EST responses and the fragmentation of EST messages [4]. The Internet-Draft has only been released in December 2018, and there are no other resources, known analyses, implementations, or information other than the Internet Draft. I also realize that I am only one person, while large companies have 10- or 20-person teams working on finding a successful implementation of the protocol. My project aims to at least add to the literature concerning EST-coaps, to develop a small proof-of-concept, and to advise future young engineers concerning any setbacks and future work needed regarding the protocol.
2 Introduction

IoT devices have become more and more pervasive in our society as demand increases. We have seen the evolution of IoT devices to popularly include televisions, audio speakers, and toasters, so-called “smart-home” objects that require human-to-machine interaction. However, tiny devices, such as sensors and actuators that require little-to-no human interaction, are becoming embedded into our everyday objects as well. These are some examples of “constrained” IoT devices, because they are physically constrained by size, weight, available power and energy[3]. Because of their limited power and memory, existing application protocols (e.g. HTTP) and existing cryptographic protocols have yet to be implemented.

CoAP is a specialized web transfer protocol specifically for constrained nodes and constrained networks, providing similar functionality as HTTP with PUT, POST, GET, and DELETE methods [5]. EST allows for basic secure enrollment on IoT devices, but only IoT devices which support HTTP and TLS. This excludes many constrained devices.

EST over secure CoAP is an IETF Internet-Draft which explains a way to bring PKI technology to constrained IoT devices. EST-coaps “transport[s] EST payloads over secure CoAP (EST-coaps) to allow low-resource constrained devices to use existing EST functionality for provisioning certificates” [4]. The goal of this project is to demonstrate a proof-of-concept of EST-coaps to show whether it is possible to achieve a working implementation of this secure protocol proposal.
3 EST-coaps Protocol Design

EST-coaps uses CoAP to transfer EST messages, where these messages closely follow the “Classical” EST design. The supported message types in EST-coaps are CA certificate retrieval, simple enrollment and re-enrollment for a CA to sign a public client identity key, Certificate Signing Request (CSR) attribute messages that inform the client of fields to include in a CSR, and server-side key generation messages to provide a private client identity key if needed [4]. In my project, I will focus on the first of such message types: requests for certificate retrieval in EST-coaps.

3.1 Discovery and URIs

The main difference between EST and EST-coaps is that the latter targets low-resource networks with small packets. Therefore, URIs in EST-coaps are shortened in order to save header space.

Table 1 demonstrates how the EST-coaps URIs are shortened compared with EST URIs. Both the clients and servers must support the new URIs [4].

<table>
<thead>
<tr>
<th></th>
<th>EST</th>
<th>EST-coaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cacerts</td>
<td>/crt</td>
<td>/crt</td>
</tr>
<tr>
<td>/csrattrs</td>
<td>/att</td>
<td>/att</td>
</tr>
</tbody>
</table>

Table 1: EST and EST-coaps URI paths

The management data are discovered by sending a GET request to “/.well-known/core” with a resource type parameter, “ace.est”. The return payload will contain the root resource of the EST resources upon success. The server must support the default /.well-known/est root resource. The EST function /cacerts, or the EST-coaps function /crt, is required by the EST-coaps protocol, among others [4]. This project in particular focuses on the EST-coaps function /crt.
3.2 Payload formats

EST-coaps sends data in simple binary for smaller payloads. I will focus only on the Content-Format for /crts. Content-Format TBD 287 can be used in place of 281 to carry a single certificate in a /crts response. The client can use the Accept option in the GET request to specify the preferred Content-Format of the response. Otherwise, the server should choose format 281 [4].

3.3 Message Bindings

EST-coaps message characteristics include: the recommendation for EST-coaps requests to be in confirmable CON messages, so that servers send an ACK with a delayed response or an ACK containing the response; inclusion of options such as Uri-Host, Uri-Port, Content-Format, etc.; having EST URLs being CoAPs-based (coaps:// rather than https://)[4].

3.4 CoAP response codes

There are many response codes in CoAP for different messages which can be found in its RFC [5]. Because this project focuses on GET messages, I will continue to specify the relevant response codes for a GET request. In CoAP, the codes 2.05 or 2.03 correspond with the HTTP response code 200. If a resource is not available to the client, response codes in EST-coaps are 2.04 and 4.04 (rather than 204 and 404 in HTTPS). For unrecognized CoAP Options, the server returns 4.02 for Bad Option. HTTP error messages 400, 423, 503, correspond with CoAP response codes 4.00, 4.03, and 5.03, respectively [4].
4 EST-coaps Client

My EST-coaps client uses a Python CoAP library with support for DTLS 1.2. The idea behind doing this is because the client is meant to demonstrate the functionality of EST-coaps as a proof-of-concept. The CoAP protocol has been defined since 2014 and implementations exist across languages and platforms [5][6]. DTLS 1.2 has also been defined since 2012 [7]. This project is not meant to implement or improve CoAP or DTLS, so therefore, Python packages are used to handle these layers of the EST-coaps protocol.

The client only implements GET requests with cacerts messages. The EST-coaps Internet-Draft provides this example for a cacerts message [4]:

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

With header fields:

```
Ver = 1
T = 0 (CON)
Options
   . . .
Payload = [Empty]
```

5 CoAPs Server

The CoAPs server for this project is simple. The server contains dummy resources in order to show a proof of concept for the EST-coaps requests. The server is using the same Python CoAPs package in order to receive requests, and replies using the CoAPs protocol and response codes discussed previously.
The EST-coaps protocol provides us with an example content response with a cert, corresponding with the example given under the previous EST-coaps client discussion [4]:

2.05 Content (Content-Format: 281)

{payload with certificate in binary format}

The example also includes header fields, where the hexadecimal representation of the payload would actually be transported in binary.

Ver = 1
T = 2 (ACK)
Code = 0x45 (2.05 Content)
Options
Payload =
3082027b06092a864886f70d010702a082026c308202680201013100300b
06092a864886f70d010701a082024e3082024a308201f0a0030201020209
009189bcdf9c99244b300a06082a8648ce3d0403023067310b3009060355
040613025553310b300906035504080c024341310b300906035504070c02
4c4131143012060355040a0c0b4578616d706c6520496e63311630140603
55040b0c0d6365727469666963174696f6e3110300e06035504030c0752
6f6f74204341310e170d3139303130373130343034315a170d3339303130
323130343034315a3067310b300906035504070c0243131143012060355040a0c0b
4578616d706c6520496e6331163014060355040b0c0d6365727469666963
6f6f74204341310e06035504030c07526f6f7420434130593031306072a
8648ce3d020106082a8648ce3d030107034204814994082b6e8185f3df
6 Considerations and Future Work

As previously mentioned, the EST-coaps protocol is only an Internet-Draft and has gone through many revisions since it came out in 2018. The only resources I could find online all cited the Internet-Draft. Most information on the protocol might be proprietary or internal to research groups and companies at the moment. I was able to learn and answer major questions about the protocol because I was able to connect with my peers at Cisco, but otherwise, the protocol is extremely new, cutting-edge, and difficult to find literature on. Future work would include more discussion by the technology community about the protocol, and more accessible literature for those less well-versed in networks and Internet Standards.

I also understand that many engineering teams across research groups and companies are working on this protocol. Therefore, I very quickly knew that I would not accomplish everything that I proposed in the original project proposal. I was able to implement a simple request, but future work would implement all EST-coaps messages.
Another point to consider is the lack of support for DTLS-secured CoAP libraries in Python 3, or in a language that I am familiar with in general, such as Java, Ruby, and others. Many CoAP libraries have some implementation of DTLS-secured CoAP, but most are works in progress and very limited. This was important for me to find for my project, because I only wanted to demonstrate a request in EST-coaps, not build three different protocols. Given the time frame of my project, I could not fully implement DTLS, CoAP, and EST-coaps on my own. Future work includes an open-source solution to integrating full security in EST-coaps.

7 Conclusion

Overall, I hope that this project can serve as a source or guide to beginning to implement EST-coaps for a young engineer. One of the greatest challenges I faced beginning this project was to gather information on a topic that there is virtually no information about. However, throughout my time gathering background information about the project and constrained IoT devices, the Internet-Draft went through many drafts, and I was able to observe the life cycle of an Internet-Draft on the Standards track.

There is significant work left to be done regarding EST-coaps. There is not yet a full implementation of the Internet-Draft or open-source proof-of-concept. I hope that my project can be expanded to include other EST-coaps messages. Hopefully, projects like these can bring the technology community one step closer to ensuring security for constrained IoT devices.
8 Acknowledgements

I would like to thank my advisor, Professor Yang, for excitedly accepting my project and for meeting with my several times throughout my time spent on the project, especially at times when I thought this completing this project would be nearly impossible. I am also incredibly grateful for my manager Steve Lang for suggesting the project to me in the first place, and software engineer Ryan Granger (both at Cisco Systems) for being an extremely valuable source of information, given that the protocol is cutting-edge, making it very difficult to find outside information.

References


