DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2018

Evaluation of Certificate Enrollment over Application Layer Security

ALEXANDROS PANAGIOTIS KRONTIRIS

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Evaluation of Certificate Enrollment over Application Layer Security

ALEXANDROS PANAGIOTIS KRONTIRIS

Master in Computer Science
Date: May 7, 2018
Supervisor: Göran Selander, **Ericsson AB**
Shahid Raza, Rikard Höglund, **RISE-SICS**
Jim Dowling, **KTH**
Examiner: Johan Håstad
Swedish title: Utvärdering av certifikatsskrivning över applikationslagersäkerhet
School of Computer Science and Communication

Abstract

This thesis analyzes Application Layer security protocols for certificate enrollment and management. EDHOC, Ephemeral Diffie-Hellman Over COSE, is a recently developed key exchange protocol which is designed to provide authentication and key-exchange functionality with compact message sizes and minimum round-trip-time. The work of this thesis extends the EDHOC protocol with a certificate enrollment functionality, targeting IoT constrained devices and it has been implemented for analysis and evaluation purposes. The main scope of this document is to study the security, performance and scalability (in descending order of importance) of enrollment over EDHOC compared to other certificate enrollment protocols.

Keywords

Certificate Enrollment/Management, application layer security, Ephemeral Diffie-Hellman Over COSE (EDHOC), key exchange protocol.

Sammanfattning

Detta examensarbete analyserar säkerhetsprotokoll av typen *Application Layer* för certifikatregistrering och hantering. EDHOC, *Ephemeral Diffie-Hellman Over COSE*, har implementerats, analyserats och utvärderats. EDHOC är ett nyligen utvecklat *Application Layer*-protokoll som är utformat för att tillhandahålla autentiserings- och nyckelfunktionsfunktioner med kompakta meddelandestorlekar och minimala rundturstider, inriktat på IoT-begränsade enheter. Huvudområdet för examensarbetet är att studera säkerhet, prestanda och skalbarhet (i fallande ordning av betydelse) hos EDHOC jämfört med andra föreslagna *Application Layer*-säkerhetsprotokoll som utför certifikatsskrivning.

Nyckelord

`Certifikat inskrivning/förvaltning, applikationslager säkerhet, Ephemeral Diffie-Hellman Over COSE (EDHOC), nyckelutbytesprotokoll.`

Contents

Chapter 1

Introduction

The Internet of Things (IoT) is a network of sensors, actuators and other electronic devices aiming to digitize our interaction with the physical world. The main element of the IoT is the data being exchanged among devices, data which is subject to analysis in a multitude of ways, *e.g.* energy consumption insights in smart homes, real-time traffic analysis, etc. Common IoT consumer applications can be found in the automotive sector, in personal health devices and smart home solutions.

The increasing applications of the IoT raises the demand for secure, lightweight and scalable security protocols. The devices of the IoT network, can be visualized as a grid, forming a virtual electronic "living" organism constantly exchanging information between its nodes. Such devices usually feature low specification hardware resources to reduce production costs, portability, along with other undesirable factors that would slow down the growth rate of the underlying market. IoT devices are growing rapidly in numbers and according to online reports [Nordrum, 2016] the number of connected devices by 2021 is estimated to be about 28 billion.

The data that such devices collect for analysis raise a serious security concern in terms of privacy which could lead to higher implications if a malicious third party were to intercept and steal such data, causing harm to individuals or group of individuals. With the continuous growth of technology we tend to find more use cases for such devices; from complete home automation solutions, to medical implants such as pacemakers and more. IoT electronic devices are more attractive to cyber attacks due to their large availability and the vast amount

as well as type of data that is being exchanged between them.

"We have taken control of your coffee pot and unless you pay $5, you won't have your coffee" , Madnick, a Professor of Information Technologies from MIT Sloan school wrote in an article published on Forbes. Putting things into perspective and taking under consideration the almost intruding rate at which electronic devices find themselves into our every day life, the possibility of such an incident occurring in a device other than a coffee maker would not have such a harmless result.

Identifying and authorizing IoT devices for the purpose of joining a network domain is a process that requires scalability and performance since the number of devices is increasing in a high rate. The Public Key Infrastructure (PKI) which is based in digital certificates is the preferred way to manage access of such devices in network domains.

At the moment, setting aside the IoT, certificate enrollment is performed by Certificate Authority (CA) vendors with the Certificate Management Protocol (CMP). Nexus is one of those vendors and offers a more automated solution by using the Simple Certificate Enrollment Protocol (SCEP) because of its wide appeal due to its simplicity despite the fact that it has not been standardized yet. Nexus [Furuhed, 2017] mention some of the disadvantages of SCEP such as its lack of support for distribution of CA generated keys and insufficient functionality for certificate renewal which led Nexus to look at the Enrollment over Secure Transport (EST) and include it in their supported enrollment protocols since it covers the aforementioned areas in which SCEP falls short and shows promising results for faster processing of the enrollment process in more complicated IoT setups. TLS, which EST uses, is an established protocol and thus an ideal candidate for enrollment because of its extensive research and development (first version was defined on January 1999). EST could also be used with Datagram Transport Layer Security (DTLS) which offers a more lightweight solution, by making use of as much as possible of TLS's implementation while at the same time making it independent of the Transport Layer, providing the possibility to run over the User Datagram Protocol (UDP). This independence of the middle layers of the web stack is a path being explored by the industry lately.

Another protocol, also independent of the Transport Layer, still currently under development is the main scope of this thesis; Ephemeral Diffie-Hellman Over COSE (EDHOC). EDHOC has been implemented

and studied for certificate enrollment usage.

1.1 Problem statement

As a consequence of the increasing popularity of IoT devices and the rapid growth of this market, there is a need for more efficient, scalable and secure protocols for enrolling devices into a network domain.

IoT devices often operate in environment setups which use multiple transport protocols which in turn creates an additional overhead to the code that such constrained devices need to have. The use of application layer security protocols for certificate enrollment would offer the advantage of making such devices independent of the various transport protocols used between different nodes of a network resulting in a smaller code footprint. Performance is related with the run-time complexity of the code, minimizing the code footprint also minimizes the system resources used to run the application and thus reduces the overall load of the device, saving additional energy. The term performance describes the efficient use of a device's resources.

In addition to performance, there are certain use cases where security over application layer is preferred instead of the traditional approach of using transport layer security. The advantage of the former is its ability to provide end-to-end encryption between client and server without requiring decryption when transporting the payload through multiple proxy servers as in some cases where TLS is used. This offers a significant motivation to research such application layer security protocols in addition to the scalability and performance aspects.

Ephemeral Diffie Hellman over COSE (EDHOC) is an application layer key exchange protocol designed to derive a session shared secret used for establishing a secure communication channel among two peers. This thesis will present a solution for certificate enrollment over EDHOC and evaluate the resulting protocol in regards to its performance and its security properties.

1.2 Research question

`What are the security properties and andvantages of an enrollment protocol over application layer security and`

how can IoT devices benefit from this? Do the benefits provide convincing arguments to promote the specification and implementation of an alternative protocol?

1.3 Acknowledgments

Chapter 2

Background

2.1 Digital certificates

A digital certificate is an electronic document used for authenticating/identifying an electronic entity for access control purposes. To put things into perspective, when a person is viewing a website from a browser, a connection exists between the browser and the server handling the website. During this connection data is being exchanged between the two parties, which usually is routed through one or more proxy servers transferring the data from one party to the other.

Intuitively, when doing tasks such as internet banking transactions the data exchanged between the user and the server is of critical importance and must be protected. Even in a seemingly insignificant everyday habit *e.g.* reading the news, the data exchanged cannot be considered as being harmless if not protected. A passive attacker would be able to extract information about the reading habits of the user by listening to the network traffic without being noticed. For this reason among others, encryption is applied in the established connection to hide the data being exchanged from third parties.

Encryption alone provides confidentiality between communicating parties. It does not provide any type of assurance for the identity of the party someone is communicating with. For the purpose of authenticating an electronic entity there is the need for some sort of electronic ID. This is the role of a digital certificate which is generated from its subject (certificate holder) information and is digitally signed by a known trusted certification authority (CA). The certificate will be used by the subject by providing it to other parties for the purpose of authentica-

tion. How this works in more detail is going to be described shortly.

2.1.1 X.509 Digital certificate structure

The International Telegraph Union Telecommunication Standardization Sector (ITU-T) defines the X.509 international standard for digital certificates. Thus when using the term certificate an X.509 formated certificate is implied.

There are three required fields for a certificate. The tbsCertificate (tbs is short for to be signed), the signatureAlgorithm and the signatureValue field [Boeyen et al., 2008, RFC 5280]. The tbsCertificate field contains the name of the subject and the certificate issuer along with the subject's public key, the validity period, the protocol's version number (currently version 3), a unique certificate serial number and other optional information. The second field contains the identifier for the cryptographic algorithm which the CA used for the signature (*e.g.* DSA with SHA-1). The third field contains the CA's digital signature, produced from the tbsCertificate, encoded as per the ASN.1 DER specification of the ITU-T.

An example of a digital certificate is shown in figure 2.1. Note that the "Fingerprints" field present in the figure is not part of the certificate. The digests under this field are outputs of the hash functions used to compute a short unique identifier, used mainly for referencing and organization purposes (or for validating EV certificates), based on the contents of the actual certificate. An example of retrieving a certificate from the web and verifying its SHA-256 fingerprint is shown in A.1; observe the length of the certificate, which is unpractical for keeping track of multiple certificates in contrast to its fingerprint.

Certificate extensions are available for a tbsCertificate in order to link additional attributes that may be required. Detailed information of all the available options and extensions of an X.509 formatted certificate is available in RFC 5280 by Boeyen et al. [2008].

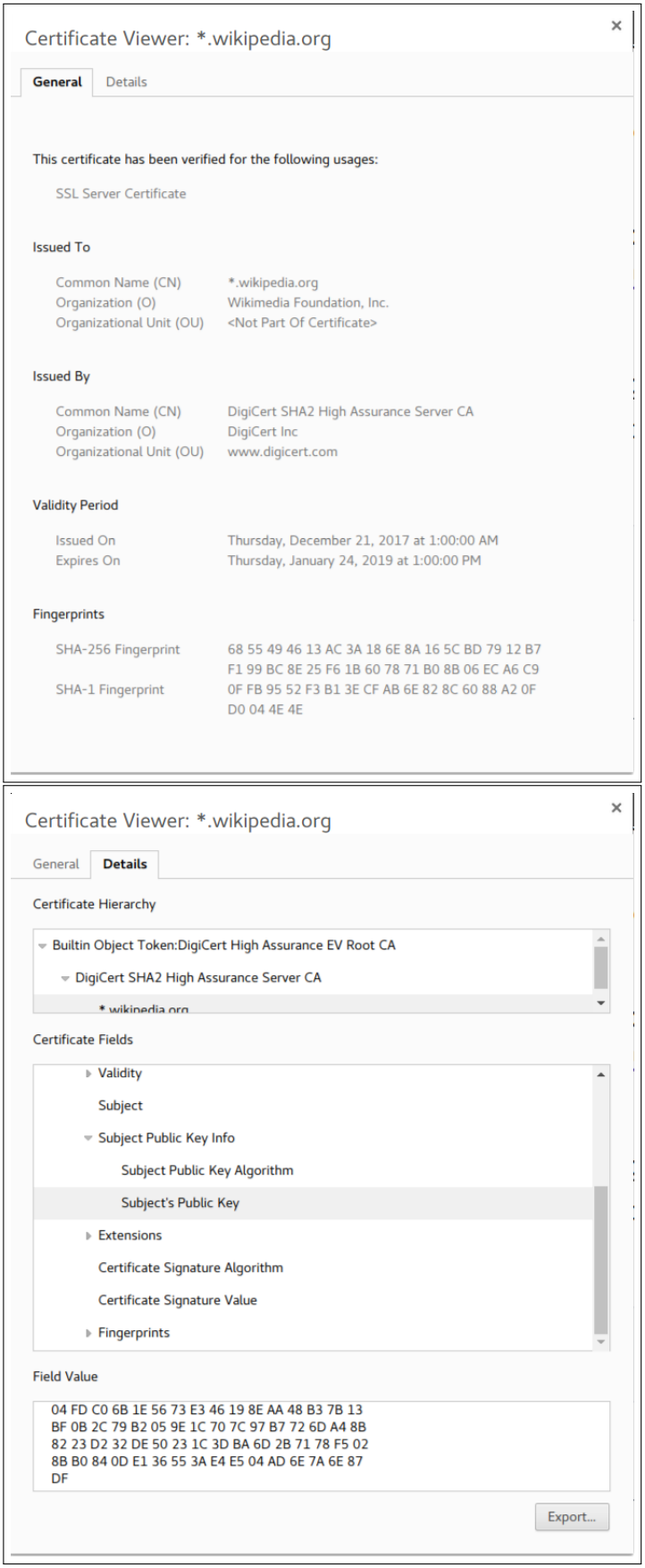


Figure 2.1: Digital certificate, courtesy of Wikipedia.

2.2 Certificate management

One of the first protocols for certificate management was the Internet X.509 Public Key Infrastructure Certificate Management Protocol by Farrell and Adams [1999, RFC 2510] (current version [Mononen et al., 2005, RFC 4210]), covering all relevant aspects of certificate creation and management by defining an infrastructure to use for this purpose. RFC 2510 utilized Public Key Infrastructure (PKI) messages which is about distributing and identifying public encryption keys.

Certificate Management over CMS (CMC) was defined by Myers, Liu, Schaad, and Weinstein [2000, RFC 2797] (current version [Myers and Schaad, 2008, RFC 5272]) intended for public key certification products and services based on CMS. The Cryptographic Message Syntax (CMS) by Housley [1999, RFC 2630] defines the syntax used to digitally sign, digest, authenticate or encrypt arbitrary messages and CMC created a set of rules for data authentication and protection using this syntax. CMC made use of the Public Key Cryptography Syntax #10 (PKCS#10) [Nystrom and Kaliski, 2000, RFC 2986] message format, presented in more detail in section 2.3.1, as the syntax used to generate a certification request. The motivation behind CMS was such so that the possibility exists to accommodate other applications that would possibly be developed in the future and make it independent of the PKCS syntax which was developed outside the Internet Engineering Task Force (IETF). Up until today the PKCS messages are used by the majority of certificate enrollment protocols.

CMC has been widely adopted by other modern certificate enrollment protocols like the Enrollment over Secure Transport (EST) and the Enrollment over Application Layer Security (EALS), but also in the legacy Simple Certificate Enrollment Protocol (SCEP). EST, EALS and SCEP are presented in section 2.5 related work.

2.3 Certificate enrollment

In order for a client to enroll and obtain a certificate it is a two-step process. First the client needs to create a certificate signing request (CSR), populate it with the required information and send it to a CA. Assuming that the client's request is valid, it will receive the generated certificate.

2.3.1 Certificate Signing Request

A certificate signing request or CSR in short, is a message sent to a CA containing the required information for obtaining a certificate. The most common formats of CSR messages are the Public Key Cryptography Standard #10 (PKCS#10), the Signed Public Key and Challenge (SPKAC), and the Certificate Request Message Format (CRMF). In this thesis the PKCS#10 format is used for the enrollment design presented in chapter 4.

A certificate signed by the CA is the response of a successful CSR. The most widely used data encapsulation formats for transporting the signed certificate to its subject are the PKCS#7 and CMS syntaxes.

PKCS#10

The Public Key Cryptography Standard #10 (PKCS#10): Certification Request Syntax Specification, defines the syntax for requesting a certificate [Nystrom and Kaliski, 2000, RFC 2986].

Breaking down the certification request, it consists primarily of three fields:

- The certification request information,
- an identifier for the signature algorithm used,
- and a signature produced from the certification request information.

The entity creating the certification request information includes its distinguished name, its public key and an optional set of attributes if there is such a need. It proceeds by signing the certification request information with its private key, the one corresponding to the certificate request subject's public key. As a final step it forms the request by including the certification request information field, the signature produced out of it and the identifier of the signature algorithm, formatting it according to the syntax that the PKCS#10 specification defines.

2.3.2 PKCS#7 and CMS

The Public Key Cryptography Standard #7 [Kaliski, 1998, RFC2315] defines a syntax for encapsulating data that could use digital signatures and digital envelopes. A digital envelope is a container of data

that is protected and should be viewed by a single or multiple specified recipients. Basically, it is data with cryptography applied to it. PKCS#7 supports the use of encryption as well as digital signatures and in addition allows the use of arbitrary attributes. Its syntax format is recursive, meaning that data can be nested within other data (*e.g.* chain of certificates).

The PKCS#7 was developed by RSA Laboratories and not inside the IETF. It was later adopted by the IETF by including it in a new document specification under the title Cryptographic Message Syntax (CMS) which provided backward compatibility (in most cases) with PKCS#7.

CMS [Housley, 2009, RFC 5652] defines the syntax for:

- Data Content Type
- Signed-data Content Type
- Enveloped-Data Content Type
- Digested-Data Content Type
- Encrypted-Data Content Type
- Authenticated-Data Content Type

PKCS#7 and CMS are used for receiving the PKI response, which is used to transport the digital certificate singed by the CA.

2.4 Enrollment for IoT constrained devices

Generally, common devices that are connected to the internet are computers or computer peripherals which are not hardware nor energy constrained. By not hardware constrained it is meant that manufacturers of such devices do not have as their first priority to cut the costs and the physical size at the bare minimum as they try to do with IoT devices.

The majority of IoT devices are constrained in a single or more of the following categories [Rayes and Salam, 2017]:

- Read Only Memory (ROM), requiring minimum code complexity from the installed firmware.

- Random Access Memory (RAM), which is required for run-time applications and other buffers that the system and user processes need to store data.
- Processing power, in the sense that the number of computing operations given a period of time are limited.
- Power resources. It is usual that IoT devices are desired to be as compact as possible and as a consequence the power source is also subject to this size reduction.
- User interface for managing software, security, applications, etc. The device should be able to operate with no or minimal user input.

A significant limitation of IoT devices is energy consumption. Networking operations can consume more than two times the power than a device would on its normal standby state [Amozarrain Perez, 2015]. In table 2.1 a comparison is presented of the current that a waspmote node draws from a battery depending on a number of different states. Of course this varies according to device specifications but the principle is the same.

Table 2.1: Waspmote device energy consumption [Amozarrain Perez, 2015]

State	Measurement
WiFi module OFF, Waspmote Standby	16mA
WiFi module OFF, Waspmote Sleep	0.0573mA
WiFi module ON, Normal	36mA
WiFi module ON, Setup	38mA
WiFi module ON, Connecting	40mA
WiFi module ON, Send	36mA

The most common security protocol, TLS, runs over TCP which is the de-facto communication protocol used in the internet. TCP besides basic data transfer offer features such as flow control, multiplexing, reliability and more [rfc, 1981, RFC 793]. The more features however, result in higher code complexity and more processor operations in general. TCP uses network buffers to store data that is to

be transmitted/received by queuing it before being processed. Each buffer is approximately 1600 bytes; let a device have one buffer to receive and one to transmit, this totals to 3200 bytes approximately. On a constrained device with about 24KB of RAM this would reserve a significant amount of memory even in a minimal buffer configuration, leaving insufficient space for system, application and other processes memory.

Most IoT devices are standalone sensors or an array of different sensors collecting data subject to analysis. If a specific measurement is lost it is not considered as significant and thus the reliability as well as other features which TCP offers do not provide any important benefit. Instead, simpler UDP based protocols are preferred, offering less overhead and minimal network activity.

Hence, an application handling its own network communication is an attractive option, since it can include only a subset of the entire TCP/IP stack functionality according to the application requirements. Such protocols can provide custom solutions keeping network communications to a minimum, require less reserved memory and offer local processing methods to achieve some of the features that TCP offers if seemed fit.

To provide a better mental conceptualization of the protocols used in IoT, below is a mapping between common protocols of the Internet stack and their counterpart in the IoT stack:

Internet Stack	**IoT Stack**
HTTP	CoAP
TLS/SSL	DTLS
TCP, UDP	UDP
IPv4, IPv6	IPv6, 6LoWPAN

- **HTTP, CoAP**: HTTP uses TCP/IP, requires synchronous communication and does not support multicast. CoAP uses UDP and

IPv6/6LoWPAN, supports multicast and is designed with less overhead due to its simplicity in comparison to HTTP.

- **TLS/SSL, DTLS**: DTLS is the TLS version over UDP. In the IoT most protocols are designed to run over UDP; DTLS provides the security for such protocols. Due to the type of data exchange in the IoT (*e.g.* sensor measurement, etc) priority is given to performance and minimal overhead in contrast to transport reliability.

- **TCP, UDP**: TCP is connection oriented, providing reliability, flow control and more. UDP is considered a "best effort" connectionless protocol which doesn't provide a reliable transport nor flow control but offers higher speed than TCP and its header is 8 bytes in comparison to TCP's which is 20 bytes. UDP also requires less messages for connection setup.

- **IPv4, IPv6, 6LoWPAN**: IPv6 is the evolution of IPv4 since the address space of the latter is not sufficient for the growing number of electronic devices. IPv6 has a global unique address system which eliminates the need for Network Address Translation (NAT) where a device would have to contact a routing server to resolve an IP address. 6LoWPAN is based on IPv6 for use with the IEEE 802.15.4 standard which defines a protocol for low-data-rate, low-power and low-complexity short-range radio frequency transmission in a Wireless Personal Area Network (WPAN) [IEEE, 2016].

2.5 Related work

Modern authentication solutions still use Pre-Shared Keys (PSK), especially in lightweight deployments, although considering the rapidly expanding IoT, PSK is not a solution that can easily scale up to demand. In most cases PSKs are configured from the factory manufacturing the device and every PSK has to be distributed among different CAs/RAs, registered and kept track of. This adds a significant overhead to the server responsible for the PSK management and any potential renewal or revocation actions.

2.5.1 SCEP

The Simple Certificate Enrollment Protocol (SCEP) [Gutmann, 2018] defines a solution for the PKI using CMS over HTTP. It is sponsored by Cisco Systems by providing client and CA implementations while also deploying it for usage.

SCEP is promoting simplicity of implementation, allowing for more straightforward interoperability, at the cost of certain functionality limitations. It't main limitation is not supporting certificate revocation transactions, forcing the client to communicate with the CA through another protocol for this purpose. SCEP does support however Certificate Revocation List (CRL) queries. In a nutshell, the general operations it supports are [Gutmann, 2018]:

- Certificate enrollment
- Certificate renewal
- Certificate queries
- Certificate Revocation List queries
- CA distributed public keys

2.5.2 EST

Enrollment over Secure Transport (EST) describes the use of TLS and HTTP to establish an authenticated and authorized communication channel for PKI clients [Pritikin, Yee, and Harkins, 2013, RFC 7030].

EST uses CMC messages for the enrollment. Messages are transported with HTTP over TLS, essentially HTTPS, and offers functionality for initial enrollment with TLS authentication of both end-entities (client and CA) through pre-installed certificates, TLS authentication without a pre-existing certificate for both end-entities and an HTTP based client authentication. In addition, it offers client certificate renewal, server generated keys and supports Full PKI Request/Response.

Because EST is based on TLS and the latter is widely deployed in the Internet for authentication, key establishment and protected communication, EST is more appealing among the rest of the enrollment protocols. With the release of TLS 1.3 which introduces major changes and optimizations in comparison to TLS 1.2, it is logical that EST will

gain more attention. Note that TLS does not try to optimize message sizes which is a big advantage for the IoT. The authors also explain that although the EST specification defines its use over HTTPS, the possibility exists that EST can also be used over Datagram Transport Layer Security (DTLS) since there are no restrictions that show it cannot.

Table 2.2: EST Layering [Pritikin et al., 2013, RFC 7030]

EST request and response messages
HTTP for message transfer and signaling
TLS for transport security
TCP for transport

EST over secure CoAP

Since the above non restrictive comment made by the authors there has been an EST variant for use over the Constrained Application Protocol (CoAP) [Shelby et al., 2014, RFC 7252] for message transportation, a protocol for constrained devices with limited hardware resources, translatable to HTTP for compatibility with the existing web infrastructure. EST over secure CoAP substitutes HTTP over TLS with CoAP over DTLS which makes possible the use of EST for constrained devices using CoAP.

2.5.3 EALS

Enrollment with Application Layer Security defines a certificate enrollment protocol utilizing IoT standardized protocols, targeting constrained devices.

EALS runs over CoAP. It formats the data according to the Concise Binary Object Representation (CBOR) [Bormann and Hoffman, 2013, RFC 7049], also targeting IoT constrained devices offering small code footprint, defining a set of extensions over the Javascript Object Notation Data Interchange Format (JSON) [Bray, 2017, RFC 8259]. CBOR

provides data serialization for EALS and the CBOR Object Signing and Encryption (COSE) [Schaad, 2017, RFC 8152] enables the use of electronic signatures and Message Authentication Codes (MAC) for authentication and encryption in parallel with CBOR.

The EALS protocol supports PKI clients to meet the scalability needs of the IoT. It defines certificate enrollment based on CMC messages like 2.5.2 and it is inspired from it, with the distinction that EALS is secured in the application layer. This is achieved with the Object Security for Constrained RESTful Environments (OSCORE) protocol [Selander et al., 2018] by providing end-to-end security for CoAP. EALS using OSCORE requires the establishment of a shared secret between the communicating entities, defined by OSCORE as the "master secret". For this purpose Ephemeral Diffie-Hellman Over COSE (EDHOC) is used. EDHOC is analyzed in detail in chapter 3. EALS is inspired from EST but is targeting IoT constrained devices and instead of using HTTP over TLS it makes use of their counterparts in the IoT protocol stack (HTTP->CoAP, TLS->OSCORE+EDHOC) to carry the CMC messages for the enrollment.

2.6 Certificate enrollment security requirements

There is no standard defining any security requirements specific to certificate enrollment. Although the obtained certificate is publicly available from the client to authenticate itself, during the process of requesting a certificate there might be certain attributes which the client might not want to leak to third parties. The most obvious scenario of such a case is when a client would request a CA generated public key, which should be transported back to the client and a potential leakage of the private key of the pair would substitute the rest of the security measures useless since at such scenario someone else would be able to pose as the legitimate client.

Below are some popular security requirements in wireless sensor networks by Zhang et al. [2009] which have been taken under consideration for the design of the enrollment protocol presented in chapter 4:

- **Availability**: The services offered should be available and not affected by DoS attacks. A common DoS attack is to flood the ser-

vice provider with messages causing the provider to be unable to process them and thus have a negative impact in the service. A good practice to mitigate such attacks is to let the client perform any cryptographic computations as well as authenticate itself before the CA does. This gives the CA the opportunity to recognize legitimate messages prior of any heavy computation and at the same time it requires a higher effort from a potential attacker to flood the CA.

- **Authorization**: The parties at each end of the channel must be authorized for the purpose of the protocol. Even if the client is authenticated it doesn't mean that it is authorized for the service the CA may provide. It is the CA's responsibility to check whether the client has the permission for the service it requests.

- **Authentication**: Each party's identity must be able to be verified by the other. Both parties must provide with proof, usually a response to a challenge which is unique to the identity per client. The most common way of authentication is by generating a digital signature on fresh random data which proves that the party generating the signature has possession of the private key it claims to have. The private key of an asymmetric key pair must be kept secret otherwise anyone knowing the private key of a party could impersonate a legitimate client.

- **Confidentiality**: The protected contents of a message must be accessible only for the party it is addressed to. Such type of protected data is encrypted using a shared secret as a key which only the communicating parties possess. It is important that in case the shared secret used for encryption is compromised, older session messages will not be compromised at the same time. This characteristic is known as Forward Secrecy.

- **Integrity**: Each party must be able to verify the data in the exchanged messages between them for possible spoofing attacks. If an active attacker alters any part of the data, both parties should be able to become aware of it.

- **Non-repudiation**: The sender of a message must not be able to deny it sent the message. Non-repudiation most often relies on

digital signatures to verify the origin of some data. If the signature of the data is verified then it is reasonably certain that the data came from the owner of the private key used for the signature. It is important that the private is kept secret, otherwise digital forgery may occur.

- **Freshness**: The messages must contain a data field holding recent data to prevent replay attacks. In security protocols freshness can be achieved by including a random nonce in the messages. The random nonce does not prove the creation date of the data, instead it makes infeasible for an attacker to re-use old messages for malicious purposes since the nonce of different session messages will be different and thus the receiving party will not be able to verify the integrity of the data. Note that to prevent delaying of messages by an attacker, freshness is implemented with a challenge-response of some sort which proves that the entity solving this challenge is recently active.

Rayes and Salam [2017] specify two additional requirement in their list of security requirements for the IoT:

- **Forward Secrecy**: A device upon leaving a network domain should not be able to access protected content exchanged within that network in the future.

- **Backward Secrecy**: A device joining a network domain should not be able to access protected content exchanged within the network belonging to prior communications.

For certificate enrollment the main concern is about data authenticity and integrity. It is important that the CSR has been requested by the entity included in the request and that this is non-reputable. The integrity of the data exchanged during the enrollment needs to be ensured to prevent malicious third parties from tampering with it.

Confidentiality is not as important because the certificate is intended to be publicly available. Although since enrollment is performed over existing communication protocols and confidentiality is a prerequisite for establishing a secure communication channel, an enrollment protocol running on top may use the encryption feature already present.

Chapter 3

EDHOC

Ephemeral Diffie-Hellman Over COSE (EDHOC) [Selander et al., 2017] is a key exchange protocol performing an authenticated Diffie-Hellman exchange for establishing a shared secret based on ephemeral keys. These keys are session specific, offer forward secrecy, and can be used by other protocols which require a shared secret such as *e.g.* OSCORE.

The encoding used in EDHOC is according to CBOR [Bormann and Hoffman, 2013] and COSE [Schaad, 2017] with the latter also providing the authenticated encryption of the protocol. Supported means of authentication are with Pre-Shared Keys (PSK), digital certificates and Raw Public Keys (RPK).

For certificate enrollment, as discussed in chapter 4 in detail, EDHOC has been picked as the communication protocol component required for the enrollment to achieve a similar outcome as in EST Pritikin et al. [2013]. The advantages of EDHOC and the reasons to perform enrollment over it, is its lightweight authenticated key exchange with small code footprint making it an ideal candidate for constrained devices as well as its size optimized messages.

3.1 COSE

CBOR Object Signing and Encryption (COSE) [Schaad, 2017] is a protocol which serves as a security extension for the CBOR data serialization format. It defines the procedures to encrypt, sign and generate MACs with CBOR. The IoT's data format of choice is CBOR and COSE builds a security infrastructure on top of it by keeping the size of the code to a minimum as well the size of the messages. EDHOC's confi-

dentiality, integrity and authentication is provided by COSE. EDHOC uses the following data objects from COSE's specification:

- (1) COSE_Sign1: COSE single signer data object.
- (2) COSE_Encrypt0: COSE single recipient data object.

(1) and (2) have counterparts for including multiple signatures as well as recipients but EDHOC does not use them since it is not intended for group communication.

3.2 Ephemeral Diffie-Hellman Over COSE

EDHOC consists of 3 messages which negotiate a set of parameters between two parties for establishing a shared secret. The security properties of EDHOC are based in the set of theoretical SIGn-and-MAc (SIGMA) protocols [Krawczyk, 2003] which describe a Diffie-Hellman key exchange offering Perfect Forward Secrecy (PFS) with digital signatures for authentication. EDHOC's supported means of authentication are pre-shared keys (PSK), raw public keys (RPK) and certificates. The credentials used for the authentication need to be established out-of-band and the method for doing this is not within the scope of the EDHOC specification. These credentials are referenced by a key ID field, KID variable in EDHOC with symmetric authentication and ID_x in EDHOC with asymmetric authentication. The public key of the asymmetric version is not included in the messages despite being public. The KID needs to be distributed ou-of-band and maintained among multiple CAs and RAs.

Let Alice and Bob be two parties that want to derive a shared secret with Alice acting as the initiator of the protocol. Bob authenticates to Alice, Alice authenticates to Bob and both parties can trust the shared secret after they have successfully authenticated to each other. Note that Bob authenticates when it sends message 2, while Alice (initiator) authenticates be sending message 3. After message 3, both parties have provided with proof to each other of successfully deriving the shared secret.

3.3 Protocol overview

Some of the message fields are different, depending on the authentication option. In the asymmetric version there is a signature inside the encrypted fields of the protocol whereas in the symmetric version there is no signature and the symmetric key is included in the method for the computation of the encryption key. Authenticated encryption is achieved in both versions. Each version is described in detail through the following sections of this chapter.

3.3.1 Asymmetric authentication

In the RPK and digital certificate authentication version the contents of each message are as shown in figure 3.1.

Table 3.1: EDHOC authenticated with asymmetric keys or digital certificate.

Party A		Party B
	$message1 \longrightarrow$	
S_A, N_A, E_A, ALG_1, APP_1		
	$\longleftarrow message2$	
S_A, S_B, N_B, E_B, ALG_2, Enc(K_2; Sig(B; ID_B, aad_2, APP_2); aad_2)		
	$message3 \longrightarrow$	
S_B, Enc(K_3; Sig(A; ID_A, aad_3, APP_3); aad_3)		

where:

- S_A is a variable length session identifier.
- N_A is a 64-bit random nonce.
- E_A is the ephemeral public key of party A.

- ALG_1 are the supported HKDF, AEAD and Sig algorithms which party A supports.
- APP_1 is the application data (optional).

Message 1

Party A generates a session identifier (for potential state recovery), a random nonce to protect against replay attacks and a session ephemeral key-pair. The session identifier, the nonce and the ephemeral public key of party A are included in message 1 along with the supported Elliptic Curve Diffie-Hellman (ECDH), HMAC Key Derivation Function (HKDF), Authenticated Encryption with Associated Data (AEAD) and signature algorithms which party A supports.

Party A is **not** authenticated to party B yet upon transmitting message 1.

Message 2

Party B generates a session identifier, a random nonce and a session ephemeral key-pair similarly to party A in message 1. Besides the session identifier, the nonce and the ephemeral public key, party B chooses an algorithm from each field (ECDH, HKDF, AEAD, Sig) and includes its choice in message 2. Party B creates a COSE_Encrypt0 object to transfer its public key identifier (ID_B) which will be used by party A to retrieve party B's public key. Optionally, application data could also be included in message 2.

By transmitting message 2, party B authenticates to party A by including its digital signature of the data in the COSE_Encrypt0 object ($Sig(...)$).

Message 3

Party A creates a COSE_Encrypt0 object and includes its public key identifier (ID_B) for party B to retrieve party A's public key. It then transmits message 3, including the COSE_Encrypt0 object and the session identifier (S_B) from message 2 which party B sent.

This is the final stage of the protocol in which party A has been authenticated to Party B and the shared secret has been successfully established and can start being used.

3.3.2 Symmetric authentication

Symmetric key authentication EDHOC messages are similar to the asymmetric version with the subtraction of the signature fields ($Sig(...)$) fields of message 2 and 3.

Table 3.2: EDHOC authenticated with Symmetric keys.

Party A		Party B
	$message1 \longrightarrow$	
	S_A, N_A, E_A, ALG_1, KID, APP_1	
	$\longleftarrow message2$	
	S_A, S_B, N_B, E_B, ALG_2, Enc(K_2; APP_2; aad_2)	
	$message3 \longrightarrow$	
	S_B, Enc(K_3; APP_3; aad_3)	

where,

- S_A is a variable length session identifier.
- N_A is a 64-bit random nonce.
- E_A is the ephemeral public key of party A.
- ALG_1 are the supported HKDF, and AEAD algorithms which party A supports.
- KID is the key ID of the PSK.
- APP_1 is the application data (optional).

Message 1

Party A generates the session identifier, random nonce and session ephemeral key-pair. It populates message 1 by including the supported HKDF and AEAD algorithms as well as the KID of the PSK used.

As in asymmetric authentication message 1, the algorithms to be used are negotiated. Note that neither party A nor B is transmitting the PSK in any message. The proof of possession is established upon successful decryption of the COSE_Encrypt0 object, of which its encryption key is dependent on the PSK and not on the standalone shared secret as its counterpart in the asymmetric version.

Message 2

Party B receives message 1 and retrieves the symmetric key referenced by the KID field. Party B generates its session identifier, the random nonce and a session ephemeral key-pair. It composes message 2 by adding the session identifier of party A it received in message 1, the self-generated session identifier, the nonce, the ephemeral public key, the HKDF and AEAD algorithm choices it made, and it creates a COSE_Encrypt0 object optionally containing any application data.

Upon reception of message 2 by party A and successful decryption of the COSE_Encrypt0 object, *i.e*. PSK proof of possession, party B is considered to be authenticated.

Message 3

Party A composes message 3 with the session identifier it received from message 2 and creates a COSE_Encrypt0 object including any optional application data.

Party A is now authenticated to party B, upon successful reception and decryption of the COSE_Encrypt0 object from party B. Now both parties are mutually authenticated and are aware they have successfully agreed on the shared secret.

3.4 Message sizes

The symmetric authentication version of the EDHOC protocol has been implemented as part of this thesis. In figure 3.3 the size of each message is listed along with the total size of all 3 messages. The sizes for the asymmetric authentication version have been theoretically calculated from the sizes of the symmetric one.

During the measurements, the application data field (APP) was not used since it is optional and the variable length session identifier

field (S_x) was fixed to 5 bytes to simulate a reasonable size which would be used in an application of the protocol.

Table 3.3: EDHOC message sizes.

Payload	Symmetric	Asymmetric
Message #1	72	70
Message #2	82	199
Message #3	22	137
Total	176	406

As seen in the above figure (3.3) the total size of the asymmetric version is $\sim$ 2.3 the size of the symmetric version. The symmetric one requires less bytes to be transmitted over an EDHOC session at the cost of managing PSKs which does not scale well as the number of devices increase due to the extra requirement for key provisioning and management.

In figure 3.4 the size of the TLS handshake messages is shown in three different scenarios. In the first scenario standalone PSK is used for authentication, in the second PSK together with ephemeral Diffie-Hellman keys and in the third scenario asymmetric key pairs are used in combination with ephemeral Diffie-Hellman keys.

Table 3.4: TLS.

Payload	PSK	PSK+DH	DH
Message #1	104	142	107
Message #2	96	135	264
Message #3	51	51	167
Total	251	328	538

Figure 3.5 displays the EDHOC message sizes depending on authentication method in the same format as used in figure 3.4 to list the size of the TLS handshake messages. EDHOC does not support authentication with standalone PSK, hence the first column is omitted.

Table 3.5: EDHOC.

Payload	PSK	PSK+DH	DH
Message #1	-	72	70
Message #2	-	82	199
Message #3	-	22	137
Total	-	176	406

3.5 Security considerations

EDHOC's security properties are inherited by the SIGMA-I [Krawczyk, 2003] theoretical protocol which it is based on. SIGMA-I offers identity protection of the initiator against malicious parties performing an active attack and identity protection of the responding party from passive attacks.

Depending on the requirements of the deployment for which EDHOC will be used, the roles of the initiator and the responder should be assigned accordingly. For example, in a client-server scenario, usually the client's identity is more important to protect and for this reason it is preferred that the client will act as the initiator.

The integrity protection in EDHOC which includes the unprotected data of a session in the encryption process as Additional Authenticated Data (AAD) is crucial for mitigation of active attacks. Lastly, the shared secret which the initiator can compute after receiving message 2 (responder can compute it by receiving message 1) should not be used until the protocol has terminated and both parties are mutually authenticated.

Chapter 4

Enrollment over EDHOC

EDHOC as described in chapter 3 is intended as a key exchange protocol for IoT applications implementing CBOR, COSE and CoAP (*e.g.* OSCORE). Since the protocol establishes a secure and authenticated channel between communicating parties it can be used for certificate enrollment with some additions which are going to be discussed throughout this chapter. For doing enrollment over EDHOC, the Certificate Management over CMS (CMC) protocol will be used which defines the use of the Cryptographic message Syntax within the scope of certificate management.

4.1 Enrollment design

The communication channel which EDHOC establishes can be utilized to accommodate the messages of the enrollment protocol. CMC is one of the most used enrollment protocols and offer a simple yet well-researched solution by utilizing the Public Key Cryptography Standards. In a nutshell, in order for a client to enroll it needs to send a certification request (CSR) in the PKCS#10 format to the CA and assuming the request is valid the client will receive a PKCS#7 response which will carry the signed certificate.

In a simple enrollment scenario there are 2 parties involved, a client device and a Certification Authority (CA). The same CA can be used by multiple clients wishing to enroll at the same time and thus the enrollment design has to take under consideration the increased load CAs will have because of the need to handle multiple concurrent connections in contrast standalone clients.

According to EDHOC's design the initiator of the protocol does not have to perform any computational expensive operation such as encryption, decryption, signing or signature verification during the generation of the first message. The receiving party of the first EDHOC message has to perform signing and encryption in the asymmetric authentication version and standalone encryption in the symmetric authentication version in order to generate message 2. Hence, giving the client the possibility to initiate EDHOC could result in the CA performing expensive operations before the client is authenticated. This is not preferable since it is more likely for an attacker to pose as a client rather than a CA. For this reason, client initiated EDHOC would help mitigate a potential denial-of-service attack which could be caused by having to generate computational expensive material before client authentication. Considering this opportunity window for a flooding attack in the certificate enrollment use case of EDHOC and the fact that the CA is not supposed to enroll devices at its own will, there are 4 design solutions.

The 1st and 2nd versions of enrollment over EDHOC are with asymmetric key (or certificate) authentication, with the client authenticating first in the 1st version and the CA authenticating first in the 2nd. The 3rd and 4th versions are with symmetric authentication, similarly, with the client authenticating first in the 3rd version and the CA authenticating first in the 4th. Each version has its pros and cons which will be analyzed throughout the rest of this chapter.

4.1.1 Certificate enrollment over EDHOC asymmetric key authentication, version #1

In this version, an asymmetric key pair or certificate is required, which is installed by the manufacturer or potentially by a trusted third party on the client device. The public key is distributed among the CAs to which the client device will request to enroll, or is stored in a central directory where a CA will be able to retrieve it when there will be the need to authenticate the client device.

The asymmetric key pair can be used either within the limited scope of initial certificate enrollment and made obsolete in which case the digital certificate obtained through the process will be used for future re-enrollment, or kept in the client device and allow for its reuse. Lastly the CA's certificate will also need to be installed on the client

device so that it will be able to authenticate the CA during the run of the enrollment protocol. The enrollment proceeds as follows:

Table 4.1: Certificate enrollment over EDHOC with asymmetric authentication, version #1.

Client (B)	CA (A)
$trigger\ message \longrightarrow$	
N_B	
$\longleftarrow EDHOC\ message\ 1$	
APP_1(N_B)	
$EDHOC\ message\ 2 \longrightarrow$	
APP_2(PKCS#10)	
$\longleftarrow EDHOC\ message\ 3$	
APP_3(PKCS#7)	

Additions to the protocol are denoted in red, application data payload in blue.

$trigger\ message \longrightarrow$

The client that wishes to enroll starts by sending a trigger message to the CA. It contains a random nonce (N_B). At this point the nonce is used to provide the ability to bind the trigger message to the rest of the messages at a later stage of the protocol.

$\longleftarrow EDHOC\ message\ 1$

The CA replies to the trigger message sent by the client with EDHOC message 1. The application data field that EDHOC defines is used to include the nonce N_B generated by the client and received by the CA in the trigger message. This binds the trigger with the EDHOC session messages since application data is included in the encryption

key derivation process as additional authenticated data as shown in section 3.2 in Selander et al. [2017].

Message 1 is not encrypted nor signed. When the client receives message 2 it is important to follow the verification processes which EDHOC defines to verify the integrity of the exchanged data.

$EDHOC\ message\ 2 \longrightarrow$

Client generates EDHOC message 2. This message authenticates the client to the CA through the signature included inside the $COSE\_Encrypt0$ object. The signature covers the public key identifier of the client ID_B, the additional authenticated data field aad_2 and the application data field APP_2.

The public key identifier is established out-of-band between the client and the CA, ideally at the same time when the manufacturer will install the asymmetric key pair or preferably the certificate on the client device. The additional authenticated data field now includes the nonce N_B which was introduced in the trigger message since it is included in the application data field of message 1 which is covered by aad_2 in EDHOC. Freshness is introduced from the signing operation which the client has to perform, containing recent generated data sent by the CA. In addition, the signature offers non-repudiation, as well as data integrity of the enrollment request present in this message. If any of the contents of the messages exchanged so far gets altered, the CA will output a different digest when computing aad_2 and the protocol will terminate. The application data field APP_2 will contain the $PKCS\#10$ certificate signing request. Confidentiality applies because the application data field is encrypted, although there is a special case which will be discussed in the Security considerations section 4.1.1 below, where there is a potential session hijacking attack vulnerability applicable to the asymmetric authentication version of EDHOC inherited from the Diffie-Hellman exchange it uses.

$\longleftarrow message\ 3$

The CA will generate EDHOC message 3 and include the response to the certification signing request using the $PKCS\#7$ syntax to transfer the digital certificate. It is the CA's responsibility for vetting the eligibility of the client to obtain the certificate and ensure it is authorized

for this purpose. Similarly, the client has the same responsibility to verify the CA it is communicating with.

Non-repudiation of the CA's response, data integrity and confidentiality are achieved similarly as in message 2.

Security considerations

The security properties of EDHOC are meant to be preserved throughout the run of the enrollment protocol regardless of the authentication method used.

A malicious third party which is more likely to target a CA than a client, motivates the above design which will results in the client authenticating itself first. The likelihood of a CA being a target in contrast to a client is not considered a significant variable since it may constantly change. In this version, the CA is able to mitigate potential denial of service attacks by raising the effort an attacker would need to make. Specifically, message 2, the generation of which requires expensive cryptographic operations, is generated by the client side. A standalone and legitimate client has to bear this cost before the CA.

However an important vulnerability is inherited by EDHOC's design which is inspired of the SIGMA [Krawczyk, 2003] theoretical protocol. This vulnerability, classified as a session-hijacking attack, is compromising the confidentiality of the application data field which EDHOC offers to transfer data. The $COSE\_Encrypt0$ objects present in EDHOC message 2 and 3 are encrypted, also containing a digital signature covering the public key identifier ID, the additional authenticated data aad and the application data APP. These fields need to be provided to the recipient of the message along with the signature in order for that party to be able to verify it. EDHOC places these fields inside the $COSE\_Encrypt0$ objects. Despite the encryption, due to the nature of the Diffie-Hellman exchange an active attacker posing as either the client or the CA will be able to derive the shared secret used for the encryption. EDHOC's goal is to establish a shared secret between two parties to secure a communication channel which will succeed EDHOC. Although, until EDHOC terminates this shared secret is not considered safe to be used because only after the end of the protocol both parties have authenticated each other. The actual computation of the shared secret precedes the time when mutual authentication is done. In more detail, the initiator of EDHOC is able to authenticate the

person it is establishing the shared secret after it has received EDHOC message 2, by decrypting the $COSE\_Encrypt0$ object of this message and verifying the signature. Similarly, the other party would authenticate the initiator, but this would take place in EDHOC message 3. An active attacker posing as the initiator would be able to establish the same shared secret and could receive EDHOC message 2 and ignore the included signature since attackers don't usually care to authenticate their victim. Vice-versa, an active attacker posing as the receiving party would establish the shared secret already from receiving EDHOC message 1 although since it would be sending EDHOC message 2 which contains the signature it doesn't compromise anything besides the data it sends itself to the victim. The principle however is that any type of data exchanged, until both parties are authenticated, cannot be considered as protected. In the process of establishing a shared secret this is fine because EDHOC will terminate if such a scenario occurs and thus a new session would be initiated. But if used in parallel to transfer application data then the aforementioned vulnerability affects the application data field present in EDHOC message 2.

4.1.2 Certificate enrollment over EDHOC asymmetric key authentication, version #2)

This version is a modification of 4.1.1 version 1 for mitigating a potential session-hijacking attack at the cost of some extra computational overhead from the CA side. Similarly to version 1 it requires an asymmetric key pair installed by the manufacturer or by a trusted third party on the client device and the distribution of the public key of this pair among the CAs to which the client device will request to enroll, or stored in a central directory where a CA will be able to retrieve it upon demand. The CA's certificate will also need to be installed on the client device so that it will be able to authenticate the CA. The certificate enrollment proceeds as follows:

Table 4.2: Certificate enrollment over EDHOC with asymmetric key authentication, version #2.

Client (A)		CA (B)
	$EDHOC\ message\ 1 \longrightarrow$	
	$\longleftarrow EDHOC\ message\ 2$	
	$EDHOC\ message\ 3 \longrightarrow$	
	APP_3(PKCS#10)	
	$\longleftarrow message\ 4$	
	S_A, Enc(K_4; aad_4, APP_4(PKCS#7))	

Additions to the protocol are denoted in red, application data payload in blue.

$EDHOC\ message\ 1 \longrightarrow$

The trigger message proposed in version 1 is not present. The client initiates a normal EDHOC session and will not request to enroll until EDHOC message 3. Thus in this method the CA authenticates to the client first. The advantages and disadvantages of this method will be discussed below in the security considerations section.

$\longleftarrow EDHOC\ message\ 2$

CA responds by generating EDHOC message 2 without any additions or alterations. The security properties of message 2 in version 4.1.1 apply and additionally the potential session hijacking attack applicable to the asymmetric authentication version of EDHOC in the previous version as discussed in 4.1.1 is also applicable to this version, although no potentially sensitive data is being leaked at this point.

$EDHOC\ message\ 3 \longrightarrow$

At this point, where the EDHOC protocol terminates, the client includes the $PKCS\_10$ enrollment request in the application data (APP_3)

field. Since both parties are mutually authenticated, with the CA being authenticated before the client and the enrollment request, the $PKCS\#10$ is essentially enveloped with the CA as the recipient.

$\longleftarrow message\ 4$

The CA after authenticating the client upon receiving message 3 by following the procedure of EDHOC, it retrieves the enrollment request and processes it accordingly. It responds with a 4th message containing the $PKCS\#7$ response. Message 4 consists of the client's session identifier S_A, and a $COSE\_Encrypt0$ object with the difference that the signature present in the $COSE\_Encrypt0$ object of message 2 and 3 is omitted since both parties are mutually authenticated and the shared secret is safe to use after the 3rd message.

The encryption key K_4 will be derived as such: It will be the session secret passed through the negotiated HKDF algorithm during EDHOC along with the following digest:

$$H(H(message1|message2)|message3)$$

H() denotes a digest and | string concatenation.

The application data field APP_4 will carry the $PKCS\#7$ response with the singed certificate. The certificate does not need to be confidential since its purpose is meant to be publicly available by its subject, although since EDHOC establishes a shared secret there is no size overhead in encrypting the $PKCS\_7$ response.

It is important to integrity protect it and bind it to the enrollment session. This is achieved from the aad field present in this message.

Security considerations

The session hijacking attack discussed in 4.1.1 is not applicable in this version because an active attacker would not gain anything. Assuming the attacker would pose as the client, initiating EDHOC, it would successfully derive a session secret which would be useless since the enrollment protocol would fail in EDHOC message 3 at the signature verification step. In the case of the attacker posing as the CA, the signature verification of EDHOC message 2 would fail and thus the protocol would terminate.

The disadvantage of this version is that the CA has to authenticate itself first by providing cryptographic material which is expensive to compute. This creates a window for a denial-of-service attack; it is trivial to generate EDHOC message 1. However, this version does not provide any motive for a malicious third party to target client devices. A manufacturer of such devices would benefit from this, but the company owning the CA would have to consider security countermeasures for mitigating the aforementioned denial-of-service attack.

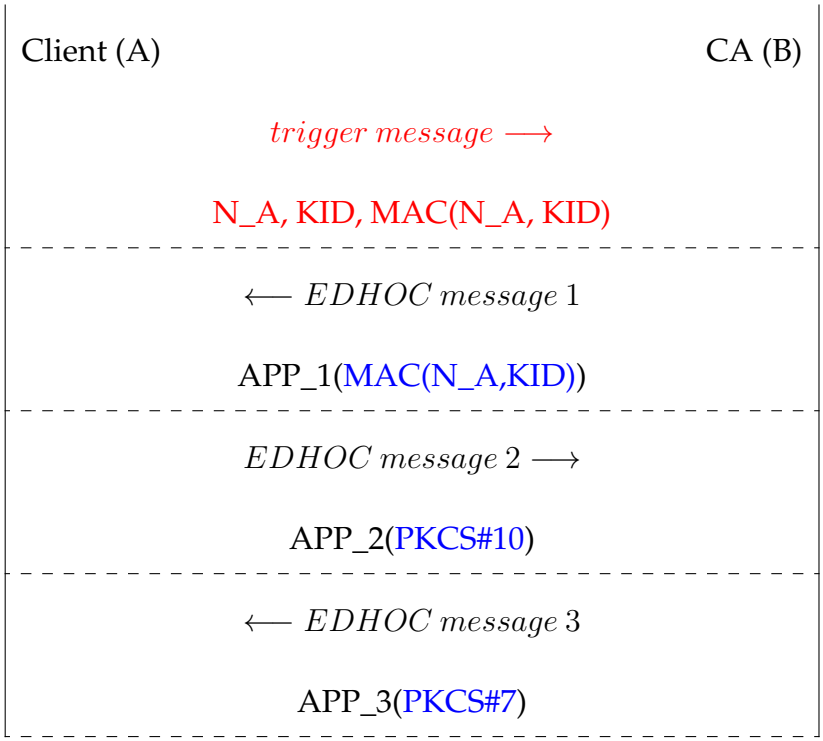
4.1.3 Certificate enrollment over EDHOC symmetric key authentication, version #1

The symmetric authentication version of certificate enrollment over EDHOC is based on an out-of-band established pre-shared symmetric key. The symmetric key is installed on the client device by the manufacturer or by a trusted third party and is also installed on the CA.

The presence of the symmetric key raises a security concern subject to its distribution among the CAs to which the client might request to enroll. The manufacturer must take care of its safe possession and distribution among the designated CAs since if compromised, any party possessing the symmetric key will be able to impersonate the identity of the client linked to this key. In addition, a certificate mandates the use of an asymmetric key pair for its generation. This asymmetric key pair in this version of certificate enrollment over EDHOC is assumed to be created and distributed by the CA, since if generated by the client then it would make sense to use the asymmetric variant from the beginning for the enrollment. However, the manufacturer may not provide support for CA generated key pairs. There could potentially be a use case where the client would generate an asymmetric key pair and share its public key with the CA while still using its symmetric key for authenticating itself. In this case there is the disadvantage of the overhead for the manufacturer, since it will have to manage and provision the symmetric key.

Having established a pre-shared symmetric key, the enrollment proceeds as follows:

Table 4.3: Certificate enrollment over EDHOC with symmetric key authentication, version #1.



Additions to the protocol are denoted in red, application data payload in blue.

trigger message $\longrightarrow$

The client initiates the enrollment by sending the trigger message. It includes a nonce N_A, the key identifier KID and a message authentication code MAC produced from the nonce and the key identifier. The nonce protects against replay attacks making it infeasible for an attacker to produce the MAC without possessing the symmetric key. The key identifier is required for the CA to identify the PSK. The client is authenticated at this early stage by the MAC it provided. Note that both for the client and the CA, computing the MAC is less computational expensive than producing/verifying a digital signature.

$\longleftarrow$ *EDHOC message* 1

CA responds with EDHOC message 1. It includes the MAC it received from the trigger message in the application data field of EDHOC APP_1 to bind the trigger message to the rest of the protocol.

Note that non-repudiation is not achieved, because multiple parties have access to the same key (*e.g.* multiple CAs and RAs). The client is able to distinguish the enrollment session with the CA, in case more than one is running, from the nonce N_A used to generate the MAC. Message 1 is not encrypted nor signed but no confidential data is included.

$EDHOC\ message\ 2 \longrightarrow$

Client generates EDHOC message 2. Unlike the asymmetric authentication version there isn't a signature present. Client authentication is achieved by including the PSK as salt to the negotiated HKDF algorithm and thus the encryption keys are dependent to the PSK. The client places the $PKCS\#10$ request in the application data field APP_2. The contents of the trigger message are also inherited (included in message 1) to the aad_2 of this message and data integrity applies. Confidentiality also applies to the PKCS#10 request because in the symmetric version the encryption key is dependent on the PSK which is not communicated to either party during the session. Due to this, the security considerations of the asymmetric version as mentioned in 4.1.1 and the scenario mentioned is not applicable in this version.

$\longleftarrow EDHOC\ message\ 3$

CA generates EDHOC message 3 and includes the $PKCS\#7$, after vetting the client, containing the response to the client's certification signing request.

Security considerations

Attention is given to preserve the security properties of EDHOC. The use of a PSK provides reduced message generation and processing times due to the lack of a signature as in its asymmetric counterpart at the cost of PSK provisioning and management from the CA owner.

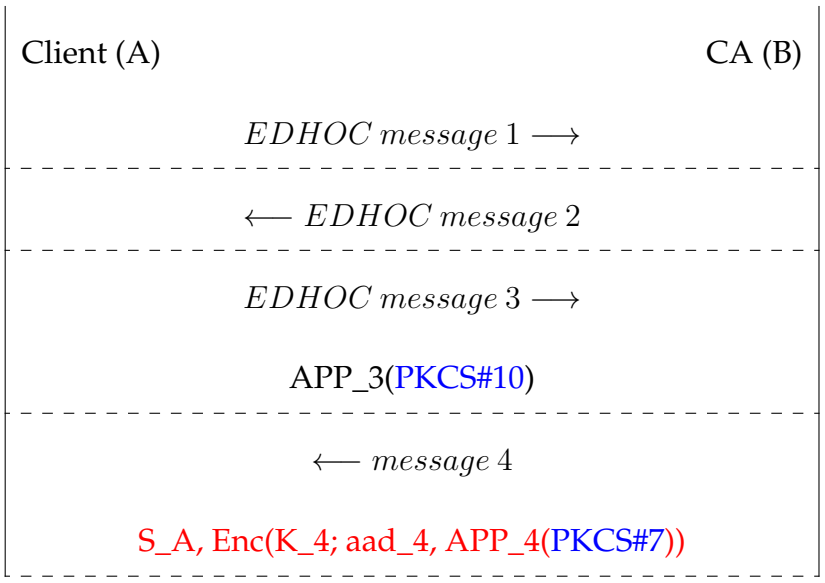
The EDHOC session shared secret in this version is secured against man-in-the-middle attacks due to its dependence to the PSK. Perfect forward secrecy applies even in case the PSK gets compromised; the attacker will not be able to decrypt previous EDHOC session messages unless one party re-uses an older session ephemeral key pair.

However, the main disadvantage of this version is the cost of handling PSKs. There are different solutions for PSK handling by CAs, with some of them maintaining a centralized storage distributing PSKs upon demand of *e.g.* a CA that has received a CSR. The leak of such a database would cause a headache to the entity responsible for redistributing new PKSs to all the devices whose keys were leaked.

4.1.4 Enrollment over EDHOC symmetric authentication, version #2

In this version the client initiates an EDHOC session and will include its $PKCS\#10$ formatted CSR in message 3 after both parties have been authenticated. The drawback of this version is the same as in the asymmetric version 2 (4.1.2); The CA will be the first to authenticate and thus will have to perform cryptographic operations before client authentication.

Table 4.4: Certificate enrollment over EDHOC with symmetric key authentication, version #2.



Additions to the protocol are denoted in red, application data payload in blue.

$EDHOC\ message\ 1 \longrightarrow$

The trigger message proposed in version 1 is not present. A normal EDHOC session is initiated by the client and the CSR is included in

EDHOC message 3.

$\longleftarrow EDHOC\ message\ 2$

CA responds by generating EDHOC message 2 without any additions or alterations. The security properties of EDHOC message 2 apply. Similarly to symmetric version 1 the vulnerability of the asymmetric version as mentioned in 4.1.1 is not applicable in this version.

$EDHOC\ message\ 3 \longrightarrow$

The client includes the $PKCS\#10$ enrollment request in the application data (APP_3) field. Both parties are mutually authenticated at this point and the $PKCS\#10$ is processed upon vetting the client.

$\longleftarrow message\ 4$

The CA responds with a 4th message containing the $PKCS\#7$ response. Message 4 consists of the client's session identifier S_A, and a $COSE\_Encrypt0$ object. The encryption key K_4 will be derived similarly as in enrollment over EDHOC with asymmetric authentication version 2 (4.1.2):

The key will be the session secret passed through the negotiated HKDF algorithm with the symmetric key as salt and the following digest:

$$H(H(message1|message2)|message3)$$

H() denotes a digest and | string concatenation.

As mentioned in 4.1.2, the certificate does not need to be confidential but because there is no size overhead in encrypting the $PKCS\#7$ response, EDHOC's $COSE\_Encrypt0$ is adopted in this version as well. The integrity protection and the binding of this message to the enrollment session is achieved from the aad field present in this message.

Security considerations

The security properties of enrollment over EDHOC with symmetric authentication version 1 (4.1.3) apply. The downside of this version is the potential window for a denial-of-service attack as discussed in 4.1.2.

This 4-message version symmetric authentication for enrollment has a benefit in comparison to its asymmetric 4-message counterpart. While in both versions the CA has to perform cryptographic computationally expensive operations before the client is authenticated, since in this version there is no signature present inside the encryption and authentication is integrated into the encryption key generation process there is no signature verification operation that needs to be performed.

Chapter 5

Results

To obtain the measurements present in this chapter, certificate enrollment over EDHOC with symmetric key authentication version #2 (as shown in 4.1.4) has been implemented. The measurements for the other versions have been theoretically calculated based on the aforementioned implemented version.

A digital certificate is linked to a key pair. The elliptic curve chosen for the key pair generation in the enrollment process is the P-256 curve which is in the recommendations list for key management by the National Institution of Standards and Technology (NIST) [Barker, 2016]. A P-256 key, which is as the name suggests 256 bits, offers the same cryptographic strength as a 3072 bit RSA key. The size of the key is important because enrollment over EDHOC is targeting IoT devices and the subject's public key as well as the signatures which are being transported throughout the enrollment session are, size wise, the biggest objects. By using bigger keys the efficiency which EDHOC provide is deemed insignificant.

5.1 Subject and CA data

A basic set of attributes is used by the subject in the CSR and by the CA generating the certificate in order to simulate an enrollment scenario. These attributes are:

For the subject:

- Country (C): **SE**

- Locality (L): **Stockholm**
- Organization (O): **Alex, Inc.**
- Common Name (CN): ***.alex.se**

For the CA:

- Country (C): **SE**
- Locality (L): **Stockholm**
- Organization (O): **Certificate Authority, Inc.**
- Common Name (CN): ***.CA.se**

In general, the information required in a certificate is set by the use-case that the certificate will be applied to and the attributes are set accordingly, in order to be able to distinguish a unique device/user. For other use-cases in which more information is used this could be done for enabling extraction of user data for a back-end system which aims to provide additional services to the device/user who is the owner of the certificate.

Note that there are cases where IoT devices do not need to use all of the above attributes if the certificate is intended for authentication in a specific domain. It could only include a unique device identifier. In IoT communication, the main purpose of the certificate may just be to enable the device to establish a secure connection to another end-point for which trusting the CA issuing the certificate is important, but does not typically require more than a unique device identifier.

5.2 Data encodings and formats

In the list below, the first two items are encodings whereas the other two are formats. These four data encodings/formats are used to present the measurements throughout this chapter. These are:

- PEM: Privacy Enhanced Mail. Base64 encoded ASCII data. This is a container format that encapsulates DER encoded certificates as well as PKCS#7, PKCS#10 and more. PEM is essentially a

base64 encoding for cryptographic data which is usually in binary form since is some cases (*e.g.* email) binary data is not supported.

- DER: Distinguished Encoding Rules. Binary encoding for ASN.1 data. It is used in situations when a unique serialized representation is required (*e.g.* cryptographic data).
- PKCS#10: See section 2.3.1.
- PKCS#7/P7B: See section 2.3.2

5.3 CSR, PKCS#7, CSR+PKCS#7

The implementation of enrollment over EDHOC uses the DER encoding for transporting both the CSR (PKCS#10) and the PKCS#7 over the network. The size in the measurements below is given both in PEM and DER. The PEM is given for comparison purposes. The attributes used as input for the certificate enrollment process are described above in 5.1

5.3.1 CSR

The size of the PKCS#10 formatted CSR generated by the subject is shown in table 5.1.

Table 5.1: Certificate Signing Request (PKCS#10) size.

Encoding	Size (in bytes)
PEM	428
DER	264

In table 5.2 the size of the PKCS#7 is shown, containing the CA generated certificate to be transported to the client.

5.3.2 PKCS#7

Table 5.2: PKCS#7 (P7B) size.

Encoding	Size (in bytes)
P7B	627
DER	432

5.3.3 CSR+PKCS#7

Finally, table 5.3 shows the size of the CSR and PKCS#7 combined.

Table 5.3: Complete enrollment size.

CSR, PKCS#7 encoding	CSR + PKCS#7 (in bytes)
PEM, P7B (Base64)	1055
DER, DER	696

5.4 Enrollment over EDHOC, trigger variant versions

The measurements presented in this section have been theoretically calculated based on the implemented version. The object header size lists only the size of the CBOR object's header and the payload lists the size of the data used in the protocol. The null character, which is required in some variables, takes up 1 byte and it is included in the payload. Potentially, some bytes could be further saved by using different CBOR object types (*e.g.* definite instead of indefinite).

Table 5.4 below, shows the size of the trigger message for both asymmetric and symmetric authentication. The MAC is produced by using the SHA-256 algorithm set to a digest size of 224 bits.

Table 5.4: Enrollment over EDHOC Trigger message

Object	Object header size	Payload (Asymmetric)	Payload (Symmetric)
MSG_ARRAY	2	0	0
N_{A,B}	1	8	8
KID*	1	-	6
MAC	1	-	28
Total	3(Asym), 5(Sym)	8	42

Table 5.5 shows the size of the complete certificate enrollment process including the CSR and the PCKS#7 formatted certificate, both for asymmetric and symmetric authentication.

Table 5.5: Enrollment over EDHOC size (trigger message versions).

	Message size + Enrollment payload size (in bytes)	
Message	Asymmetric	Symmetric
Trigger message	11 + 0	47 + 0
EDHOC message 1	70 + 11	72 + 11
EDHOC message 2	199 + 269	82 + 269
EDHOC message 3	137 + 437	22 + 437
Total	1134	940

5.5 Enrollment over EDHOC, 4-message variant versions

In this section the measurements of of the symmetric authentication version have been obtained from the implementation. The measurements for the asymmetric authentication version have been calculated based on the symmetric ones.

Similarly to 5.4 the object header size lists only the header of the CBOR objects and the payload lists the size of the data used for the protocol. Potential size optimizations are also possible in this variant as well.

Table 5.6 shows the size of the 4th message of the enrollment over EDHOC protocol. Note that the size of the PKCS#7 formatted certificate which is transported to the client in this message is not included in the payload measurements.

Table 5.6: Enrollment over EDHOC Message #4

Object	Object header size	Payload (Asymmetric)	Payload (Symmetric)
MSG_ARRAY	2	0	0
S_A*	1	5	5
COSE_Encrypt0	2		
protected	1	0	0
unprotected	1	0	0
body	1	8	8
array	1	0	-
protected	1	37	-
unprotected	1	0	-
body	1	8	-
signature	1	65	-
Total	13(Asym), 8(Sym)	123	13

In table 5.7, the complete size of enrollment over EDHOC is shown including the CSR and the PKCS#7 object sizes, each of which has been added to its corresponding message (CSR -> EDHOC message 3, PKCS#7 -> Message 4).

Table 5.7: Enrollment over EDHOC size (4-message versions).

	Message size + Enrollment payload size (in bytes)	
Message	Asymmetric	Symmetric
EDHOC message 1	70 + 0	72 + 0
EDHOC message 2	199 + 0	82 + 0
EDHOC message 3	137 + 269	22 + 269
Message 4	136 + 437	21 + 437
Total	1248	903

5.6 Memory usage

Memory usage measurements are presented in this section of the enrollment over EDHOC with symmetric authentication version #2. In IoT constrained devices memory usage is important since such devices can have very limited amount of RAM.

The tool used for the measurements is `memusage`. It is a script for Linux written in bash which profiles the memory usage of an executable. It calculates the allocated memory by intercepting the calls made to `malloc()`, `realloc()`, `calloc()` and `free()`, which are the functions used for the memory allocation operations in the implementation (written in C). More information about `memusage` can be found at the Linux user manual [memusage Linux manual].

5.6.1 OpenSSL cli CSR generation

As a point of reference, the memory usage of generating a CSR using the OpenSSL client from the command line in a Linux terminal was measured. Figure 5.1 lists the results and figure 5.2 shows the data plotted in a graph. These measurements represent the minimum memory resources required by OpenSSL's program to generate the CSR.

```
Memory usage summary: heap total: 162716, heap peak: 95922, stack peak: 5440
________total calls   total memory   failed calls
 malloc|        4041         152028              0
realloc|          43          10688              0  (nomove:1, dec:0, free:0)
 calloc|           0              0              0
   free|        4426         161692
```

Figure 5.1: OpenSSL CSR generation memory usage.

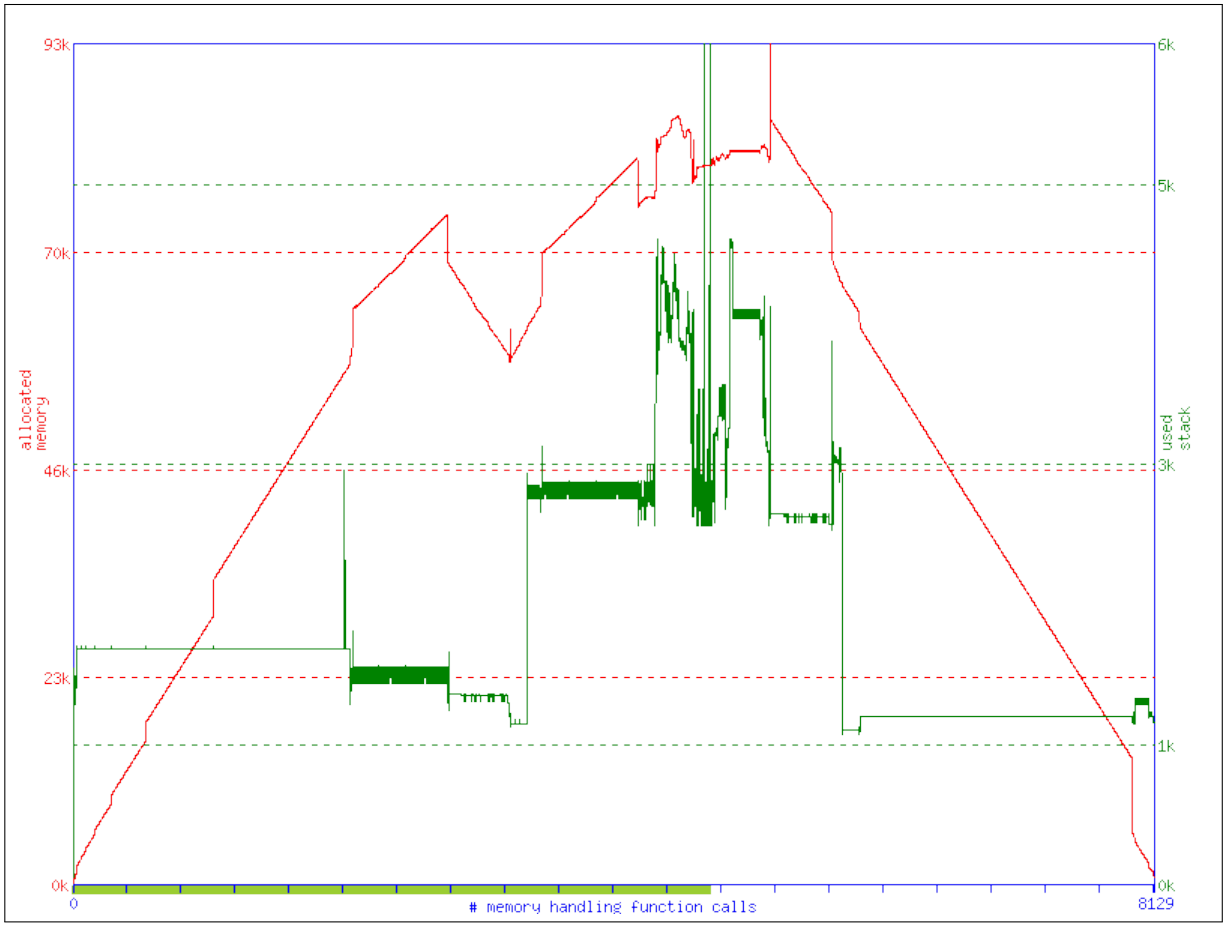


Figure 5.2: OpenSSL CSR generation memory usage graph.

5.6.2 Enrollment over EDHOC with symmetric authentication, version #2

The implementation of enrollment over EDHOC with symmetric authentication version #2 consists of a client and a server program. The

client is the subject who wishes to enroll and the server is the CA which responds to the enrollment request.

In a real-word application the CA doesn't have any hardware constraints. In fact, CAs are big servers which are able to handle multiple concurrent connections and serve multiple enrollment requests simultaneously. For this reason only the client's program memory usage was measured, since this program is going to run in constrained devices. The client does the following functions:

- Generates EDHOC message 1 and sends it.
- Receives EDHOC message 2 and parses it.
- Generates EDHOC message 3 (including the CSR) and sends it.
- Receives Message 4 (carries the signed certificate) and parses it.

It is important to note that the client program used to measure its memory resources is a proof-of-concept implementation. This means that it is **not** optimized in any way and it includes a lot of functions used for debugging which also require memory. Despite this fact, it should provide an initial insight on the memory resources the protocol requires. If optimized, the program should require at least 15% less allocated memory, although this is an educated guess. Figure 5.3 lists the results from `memusage` and figure 5.4 shows the graph of the plotted data.

```
Memory usage summary: heap total: 626259, heap peak: 196349, stack peak: 4544
________total calls   total memory   failed calls
 malloc|        6383         612187              0
realloc|          76          13488              0  (nomove:31, dec:0, free:0)
 calloc|          20            584              0
   free|        6029         513794
```

Figure 5.3: Enrollment over EDHOC with symmetric authentication verison #2, client, memory usage.

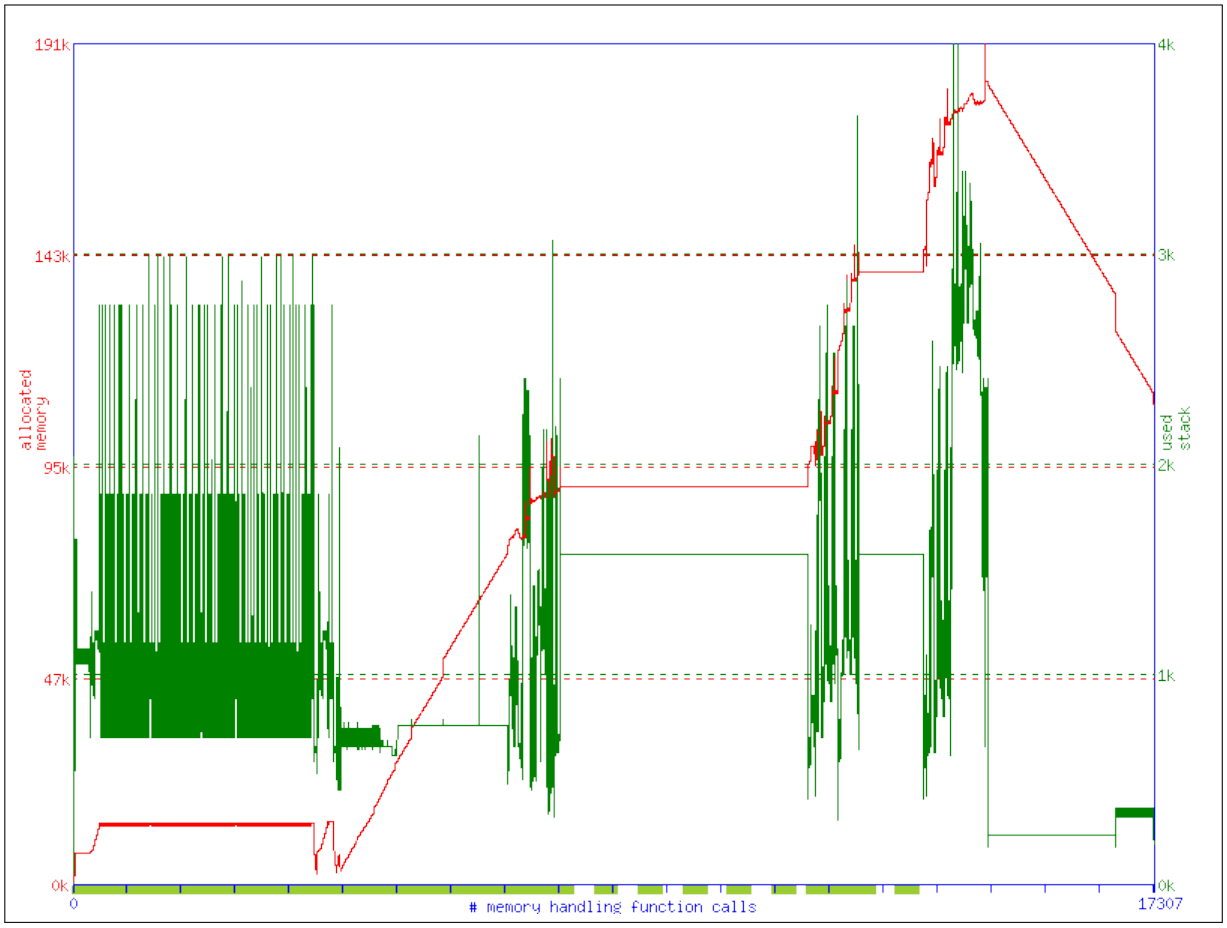


Figure 5.4: Enrollment over EDHOC with symmetric authentication version #2 memory usage.

Chapter 6

Conclusion

Certificate enrollment over application layer security offers an alternative to the defacto way of using the transport layer security protocol and the overhead that comes with its extended set of functions. IoT constrained devices cannot afford, in most cases, to use transport layer security and thus application layer provides an attractive option.

This thesis addresses the problem of enrolling devices to a network domain as the number of these devices grows over time. This is a problem which has gained attention due to the popularity of IoT devices and the need for a more scalable enrollment solution. Protocols like SCEP, EST and EALS have been designed, aiming to fill in the gap in certificate enrollment protocols. SCEP and EALS are independent of the transport layer and target deployment environments where there is a need to keep radio transmission and network buffers to a minimum; radio is the most energy hungry module of an IoT constraint device. As for EST, its main selling point is its "weakness" at the same time, if we compare it to the others, which is that it uses the TLS protocol. TLS is the most widely deployed protocol, it is well researched and hence performing enrollment over it seems logical, since it is already widely supported.

EDHOC, the protocol implemented for the purpose of this thesis, was presented and discussed in chapter 3. Its designated use as a lightweight application layer key-exchange protocol for deriving a shared secret between communicating parties fills a gap which, as of the time of writing this thesis, exists in the IoT protocol stack. The size of the EDHOC messages as presented in 3.3 show the lightweight character of the protocol. EDHOC's symmetric authentication version

offers a $\sim 46\%$ decrease in the total size of the messages exchanged during the run of the protocol in comparison to TLS when used for the same purpose. In the asymmetric authentication version this decrease in favor of EDHOC against TLS is $\sim 24\%$. Its minimalism approach results in a small code footprint and eliminates the problem of using network buffers in an IoT device which is among the main arguments against using TLS.

In this thesis, EDHOC is used to study its use for certificate enrollment. For each of EDHOC's authentication version, two different design solution are presented each with its own advantages and disadvantages.

For the asymmetric authentication version the first solution included a trigger message from the client side for initiating the protocol. Upon authenticating itself in EDHOC message 2, the client would include the CSR in the application data field of this message and receive the certificate in the application data field of EDHOC message 3 in which the CA would authenticate to the client. The second solution instead of using a trigger message, initiates the protocol directly with EDHOC message 1. The CSR is included in EDHOC message 3 (CA has authenticated itself to the client in EDHOC message 2) and the client receives the certificate in an additional 4th message. The same design solutions are presented for the symmetric authentication version of EDHOC as well, with the difference of the contents of the trigger message which instead of including a standalone nonce as in the asymmetric one, it includes the KID and a MAC covering the nonce+KID.

Of the two authentication versions we observed that the symmetric one offered size savings of $\sim 27\%$ (4-message variant) and $\sim 17\%$ (trigger variant) of the total size of a run of the enrollment protocol in comparison to the asymmetric; this difference is accounted for the lack of the signature fields. Despite the advantage of a reduced size, using symmetric authentication for certificate enrollment is not a scalable solution since the CA would have to keep track of multiple PSKs and most importantly the non-repudiation property of the enrollment protocol would not be applicable since the PSK is shared between the two parties. The asymmetric authentication doesn't have this disadvantage, however the use of digital signatures brings extra computational cost to perform the cryptographic operations for producing and verifying the signatures. Since attention is given to data integrity and authenticity when enrolling a client, the asymmetric version is the su-

perior choice from this aspect.

As for the design solution of choice per authentication version, the trigger message design offers size savings of $\sim 9\%$ in comparison to the 4-message design in the asymmetric authentication version and an increase of $\sim 4\%$ in the symmetric version due to the extra contents of the trigger message specific to the design solution of this version. Setting aside the size criterion, there are some pros and cons for each solution. The trigger design makes the client authenticate before the CA at the cost of potentially compromising the information included in the CSR (not applicable to symmetric authentication) since it is sent before both parties are mutually authenticated and at that time the client doesn't know with whom it is establishing a secure channel. In the 4-message design this is mitigated by placing the CSR in EDHOC message 3 essentially when the client is authenticating to the CA. The con of this solution is that the CA is going to be the first to perform cryptographic operations which are required for the generation of EDHOC message 2. The security properties of each design are discussed throughout chapter 4.

Enrollment over EDHOC is an ideal protocol for IoT constrained devices. The results show exactly how lightweight it is and its independence of the transport layer is an advantage which is valued in protocols of the IoT stack. The biggest limitation of constrained devices is energy and the amount of data that needs to be transmitted by a protocol between parties determines the time which the radio of the devices will have to operate for and this directly translates to energy costs since the radio is the most power hungry component of such devices.

For future work, the most interesting subject would be to perform a complete benchmark of Enrollment over EDHOC with EST followed by a security analysis of both protocols. By testing them in a real-world deployment, observations can be made of the benefits and drawbacks each protocol offers. Recently, research has been made regarding the use of compressed certificates in TLS. Ghedini and Vasiliev [2018] propose a way to compress certificate chains and reduce the data transmitted over the network. The authors observed, as also observed in enrollment over EDHOC, that the majority of the transmitted bytes are occupied by the certificates which the protocol carries. Naturally, researching certificate compression in enrollment over EDHOC is another future task, potentially yielding significant size reductions, which would

highlight the lightweight character of the protocol.

Chapter 7

Sustainability and ethics

The objective of this thesis was to research and evaluate a certificate enrollment solution for use over application layer. Directly there are no sustainability and ethic concerns applicable.

Such security protocols are used both by client and server machines and thus are indirectly responsible for the power consumption related to the cryptographic operations which they require for functioning. These operations include signature generation, signature verification, encryption, decryption and hashing. Despite the fact that the set of the cryptographic functions is fixed (open standards) and used between security protocols, each one uses such cryptographic operations differently according to its design and the features it offers. Since enrollment over EDHOC is designed to keep computationally expensive operations at a minimum, this translates to minimum power consumption on both the client and the server machines.

From an ethical point of view, there are no concerns since the purpose of security protocols like the one present in this thesis are to protect the data of the users from third parties who have malicious intentions. Hence such protocols are deployed to mitigate unethical use of the data which they transport/protect.

Bibliography

Amy Nordrum. Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated, August 2016. URL https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated.

Stuart Madnick. Security Surprises Arising from the Internet of Things (IoT). URL https://www.forbes.com/sites/ciocentral/2017/05/08/security-surprises-arising-from-the-internet-of-things-iot/.

Martin Furuhed. Support for certificate enrollment protocol EST, January 2017. URL https://www.nexusgroup.com/blog/support-certificate-enrollment-protocol-est/.

Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and Dave Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008. URL https://rfc-editor.org/rfc/rfc5280.txt.

Stephen Farrell and Dr. Carlisle Adams. Internet X.509 Public Key Infrastructure Certificate Management Protocols. RFC 2510, March 1999. URL https://rfc-editor.org/rfc/rfc2510.txt.

Tero Mononen, Tomi Kause, Stephen Farrell, and Dr. Carlisle Adams. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210, September 2005. URL https://rfc-editor.org/rfc/rfc4210.txt.

Michael Myers, Xiaoyi Liu, Jim Schaad, and Jeff Weinstein. Certificate management messages over cms. RFC 2797, RFC Editor, April

2000. URL http://www.rfc-editor.org/rfc/rfc2797.txt. http://www.rfc-editor.org/rfc/rfc2797.txt.

Michael Myers and Jim Schaad. Certificate Management over CMS (CMC). RFC 5272, June 2008. URL https://rfc-editor.org/rfc/rfc5272.txt.

Russ Housley. Cryptographic Message Syntax. RFC 2630, June 1999. URL https://rfc-editor.org/rfc/rfc2630.txt.

Magnus Nystrom and Burt Kaliski. PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986, November 2000. URL https://rfc-editor.org/rfc/rfc2986.txt.

Burt Kaliski. PKCS #7: Cryptographic Message Syntax Version 1.5. RFC 2315, March 1998. URL https://rfc-editor.org/rfc/rfc2315.txt.

Russ Housley. Cryptographic Message Syntax (CMS). RFC 5652, September 2009. URL https://rfc-editor.org/rfc/rfc5652.txt.

Ammar Rayes and Samer Salam. *Internet of Things From Hype to Reality: The Road to Digitization*. Springer International Publishing, Cham, 2017. ISBN 3319448587.

Ugaitz Amozarrain Perez. Low Power WiFi: a study on power consumption for Internet of Things. Master's thesis, Universitat Politècnica de Catalunya, 2015.

Transmission Control Protocol. RFC 793, September 1981. URL https://rfc-editor.org/rfc/rfc793.txt.

IEEE. Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, April 2016. doi: 10.1109/IEEESTD.2016.7460875.

Peter Gutmann. Simple Certificate Enrolment Protocol. Internet-Draft draft-gutmann-scep-09, Internet Engineering Task Force, February 2018. URL https://datatracker.ietf.org/doc/html/draft-gutmann-scep-09. Work in Progress.

Max Pritikin, Peter Yee, and Dan Harkins. Enrollment over Secure Transport. RFC 7030, October 2013. URL https://rfc-editor.org/rfc/rfc7030.txt.

Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014. URL https://rfc-editor.org/rfc/rfc7252.txt.

Carsten Bormann and Paul E. Hoffman. Concise Binary Object Representation (CBOR). RFC 7049, October 2013. URL https://rfc-editor.org/rfc/rfc7049.txt.

Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, December 2017. URL https://rfc-editor.org/rfc/rfc8259.txt.

Jim Schaad. CBOR Object Signing and Encryption (COSE). RFC 8152, July 2017. URL https://rfc-editor.org/rfc/rfc8152.txt.

Göran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. Object Security for Constrained RESTful Environments (OSCORE). Internet-Draft draft-ietf-core-object-security-08, Internet Engineering Task Force, January 2018. URL https://datatracker.ietf.org/doc/html/draft-ietf-core-object-security-08. Work in Progress.

Yan Zhang, Jun Zheng, and Honglin Hu, editors. *Security in wireless mesh networks*. Wireless networks and mobile communications. CRC Press, Boca Raton, FL, 2009. ISBN 978-0-8493-8250-5. OCLC: ocn122525801.

Göran Selander, John Mattsson, and Francesca Palombini. Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-selander-ace-cose-ecdhe-07, Internet Engineering Task Force, July 2017. URL https://datatracker.ietf.org/doc/html/draft-selander-ace-cose-ecdhe-07. Work in Progress.

Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc'approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Annual International Cryptology Conference*, pages 400–425. Springer, 2003.

Elaine Barker. Recommendation for Key Management Part 1: General. Technical Report NIST SP 800-57pt1r4, National Institute of Standards and Technology, January 2016. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

memusage Linux manual. memusage(1) - Linux manual page. URL http://man7.org/linux/man-pages/man1/memusage.1.html.

Alessandro Ghedini and Victor Vasiliev. Transport Layer Security (TLS) Certificate Compression. Internet-Draft draft-ietf-tls-certificate-compression-03, Internet Engineering Task Force, April 2018. URL https://datatracker.ietf.org/doc/html/draft-ietf-tls-certificate-compression-03. Work in Progress.

Appendix A

Certificate fingerprint

```
$ echo -n | openssl s_client -connect wikipedia.org:443 | sed -n
e '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > ./wikipedia.pem

$ cat ./wikipedia.pem
-----BEGIN CERTIFICATE-----
MIIIfDCCB2SgAwIBAgIQCDCUYtH+pgrgur/174vFRTANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMS8wLQYDVQQDEyZEaWdpQ2VydCBTSEEyIEhpZ2ggQXNz
dXJhbmNlIFNlcnZlciBDQTAeFw0xNzEyMjEwMDAwMDBaFw0xOTAxMjQxMjAwMDBa
MHkxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T
YW4gRnJhbmNpc2NvMSMwIQYDVQQKExpXaWtpbWVkaWEgRm91bmRhdGlvbiwgSW5j
LjEYMBYGA1UEAwwPKi53aWtpcGVkaWEub3JnMFkwEwYHKoZIzj0CAQYIKoZIzj0D
AQcDQgAE/cBrHlZz40YZjqpIs3sTvwssebIFnhxwfJe3cm2ki4Ij0jLeUCMcPbpt
K3F49QKLsIQN4TZVOuTlBK1uem6H36OCBdIwggXOMB8GA1UdIwQYMBaAFFFo/5Cv
Agd1PMzZZWRiohK4WXI7MB0GA1UdDgQWBBRurRGx7mcc61Td8ipmVMG+0DsoOTCC
AvgGA1UdEQSCAu8wggLrgg8qLndpa2lwZWRpYS5vcmeCDXdpa2lwZWRpYS5vcmeC
ESoubS53aWtpcGVkaWEub3JnghQqLnplcm8ud2lraXBlZGlhLm9yZ4INd2lraW1l
ZGlhLm9yZ4IPKi53aWtpbWVkaWEub3JnghEqLm0ud2lraW1lZGlhLm9yZ4IWKi5w
bGFuZXQud2lraWllZGlhLm9yZ4INbWVkaWF3aWtpLm9yZ4IPKi5tZWRpYXdpa2ku
b3JnghEqLm0ubWVkaWF3aWtpLm9yZ4INd2lraWJvb2tzLm9yZ4IPKi53aWtpYm9v
a3Mub3JnghEqLm0ud2lraWJvb2tzLm9yZ4IMd2lraWRhdGEub3Jngg4qLndpa2lk
YXRhLm9yZ4IQKi5tLndpa2lkYXRhLm9yZ4IMd2lraW5ld3Mub3Jngg4qLndpa2lu
ZXdzLm9yZ4IQKi5tLndpa2luZXdzLm9yZ4INd2lraXF1b3RlLm9yZ4IPKi53aWtp
cXVvdGUub3JnghEqLm0ud2lraXF1b3RlLm9yZ4IOd2lraXNvdXJjZS5vcmeCEcou
d2lraXNvdXJjZS5vcmeCEioubS53aWtpc291cmNlLm9yZ4IPd2lraXZlcnNpdHku
b3JnghEqLndpa2l2ZXJzaXR5Lm9yZ4ITKi5tLndpa2l2ZXJzaXR5Lm9yZ4IOd2lr
aXZveWFnZS5vcmeCECoud2lraXZveWFnZS5vcmeCEioubS53aWtpdm95YWdlLm9y
Z4IOd2lrdGlvbmFyeS5vcmeCECoud2lrdGlvbmFyeS5vcmeCEioubS53aWt0aW9u
YXJ5Lm9yZ4IXd2lraW1lZGlhZm91bmRhdGlvbi5vcmeCGSoud2lraW1lZGlhZm91
bmRhdGlvbi5vcmeCGyoubS53aWtpbWVkaWFmb3VuZGF0aW9uLm9yZ4ISd21mdXNl
cmNvbnRlbnQub3JnghQqLndtZnVzZXJjb250ZW50Lm9yZ4IGdy53aWtpMA4GA1Ud
DwEB/wQEAwIHgDAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwdQYDVR0f
BG4wbDA0oDKgMIYuaHR0cDovL2NybDMuZGlnaWNlcnQuY29tL3NoYTItaGEtc2Vy
dmVyLWc2LmNybDA0oDKgMIYuaHR0cDovL2NybDQuZGlnaWNlcnQuY29tL3NoYTIt
aGEtc2VydmVyLWc2LmNybDBMBgNVHSAERTBDMDcGCWCGSAGG/WwBATAqMCgGCCsG
AQUFBwIBFhxodHRwczovL3d3dy5kaWdpY2VydC5jb20vQ1BTMAgGBmeBDAECAjCB
gwYIKwYBBQUHAQEEdzB1MCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5kaWdpY2Vy
dC5jb20wTQYIKwYBBQUHMAKGQWh0dHA6Ly9jYWNlcnRzLmRpZ2ljZXJ0LmNvbS9E
aWdpQ2VydFNIQTJIaWdoQXNzdXJhbmNlU2VydmVyQ0EuY3J0MAwGA1UdEwEB/wQC
MAAwggEGBgorBgEEAdZ5AgQCBIH3BIH0APIAdwC72d+8H4pxtZOUI5eqkntHOFeV
CqtS6BqQlmQ2jh7RhQAAAWB6Rb/PAAAEAwBIMEYCIQCYBInG8WHe18O9vaEPnHxH
LYRqOcc7wiG6SCybdpzCTQIhAIgDyjxagryxHN2Y4vraBjPd/OsIi5KWNmf7uXph
IQ6gAHcAh3W/51l8+IxDmV+9827/Vo1HVjb/SrVgwbTq/16ggw8AAAFgekXAKAAA
BAMASDBGAiEAxInIUUoY+UKfTiQfcQt0Bq+RV/oKF6Qvivhful45z3wCIQCUgFZz
FpBXRl1puoXQ8S379GmZm6h8pDEPnrFymBix1DANBgkqhkiG9w0BAQsFAAOCAQEA
Cn7jA1gpa8XvPlZoaYjHBLdcBEAJDKS4at1GHvaXrEGN6wc4tURSlC8J02SV9t9I
QmQ/yVwk4OYzGBqWAqCDNkvCEblead7ENhWEUdGqVzljT2Pjp2KUtHHLTITmEY2l
GGY7amjEJcyJpxaEbZW8OBoiXajz7DlIl+Inh0mYtAtQl3QK6dnBPRyKBItRvfSy
AjL+a6v0Ad4OHuTjbYpXKRHgKu9ewVxjP2NE668rLy2bCXCdw1H3KX7NtsPHlLW4
QOtJM8xSDR4B5/V51y6yxilSHgv9rNouLc+7JChE+5aS74OlqwyMIqHg7Tuhbxt9
w6L7TYVnAT0Usyda5d06Ew==
-----END CERTIFICATE-----

$ openssl x509 -noout -in ./wikipedia.pem -fingerprint -sha256
SHA256 Fingerprint=68:55:49:46:13:AC:3A:18:6E:8A:16:5C:BD:79:12:
B7:F1:99:BC:8E:25:F6:1B:60:78:71:B0:8B:06:EC:A6:C9
```

Figure A.1: Wikipedia certificate fingerprint verification

Appendix B

EDHOC measurements

Tables B.1, B.2 and B.3 show the size of each component of the EDHOC messages as well as the size of the CBOR headers of each object. Note that the CBOR headers might be different for two identical object types because the headers also depend on the value which the object holds. For example, HKDFs_U and AEADs_U are both CBOR definite array objects but the former holds a negative value which requires one additional byte.

The objects marked with * are variables of which the length is variable and is set according to the requirements of the parties running EDHOC. The session identifier fields S_U and S_V was set to 5 bytes while the key identifier KID was set to 6 bytes to represent the potential values of these variables in a real deployment.

The ECDH-Curves_U, HKDFs_U, AEADs_U, SIGs_V and SIGs_U fields in message 1 are arrays which list the per category supported algorithms for party V to choose from. The payload size of message 1 in table B.1 is calculated by listing only one supported algorithm per category.

The optional application data fields APP_1, APP_2 and APP_3 are not included. The total size of each message per authentication version is calculated as the sum of the object header size of the corresponding version and the payload ($Sum(ObjectHeader + Payload)$).

Table B.1: EDHOC Message #1

Object	Object header size	Payload (Asymmetric)	Payload (Symmetric)
MSG_ARRAY	2		
MSG_TYPE	0	1	1
S_U*	1	5	5
N_U	1	8	8
E_U	2	38	38
ECDH-Curves_U	1	1	1
HKDFs_U	2	1	1
AEADs_U	1	1	1
SIGs_V	1	1	-
SIGs_U	1	1	-
KID*	1	-	6
Total	13(Asym), 11(Sym)	57	61

Table B.2: EDHOC Message #2

Object	Object header size	Payload (Asymmetric)	Payload (Symmetric)
MSG_ARRAY	2		
MSG_TYPE	0	1	1
S_U*	1	5	5
S_V*	1	5	5
N_V	1	8	8
E_V	1	37	39
HKDF_V	2	1	1
AEAD_V	1	1	1
SIG_V	1	1	-
SIG_U	1	1	-
COSE_Encrypt0	2		
protected	1	0	0
unprotected	1	0	0
body	1	8	8
array	1	0	-
protected	1	37	-
unprotected	1	0	-
body	1	8	-
signature	1	65	-
Total	21(Asym), 14(Sym)	178	68

Table B.3: EDHOC Message #3

Object	Object header size	Payload (Asymmetric)	Payload (Symmetric)
MSG_ARRAY	2	0	0
MSG_TYPE	0	1	1
S_V*	1	5	5
COSE_Encrypt0	2		
protected	1	0	0
unprotected	1	0	0
body	1	8	8
array	1	0	-
protected	1	37	-
unprotected	1	0	-
body	1	8	-
signature	1	65	-
Total	13(Asym), 8(Sym)	124	14

TRITA EECS-EX-2018:632