Master of Science Thesis in Secure System
Department of Electrical Engineering, Linköping University, 2017

End-to-end security enhancement of an IoT platform using object security

Hampus Tjäder

Master of Science Thesis in Secure System

End-to-end security enhancement of an IoT platform using object security

Hampus Tjäder

LiTH-ISY-EX--17/5053--SE

Supervisor: **Jingcheng Zhang**
Ericsson Research
Patrik Salmela
Ericsson Research
Mikael Asplund
IDA, Linköpings universitet

Examiner: **Jan-Åke Larsson**
ISY, Linköpings universitet

Division of Information Coding
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Sammanfattning

”Internet of Things” ses som en av de närmsta internetomvandlingarna. Inom en snar framtid kommer majoriteten av alla anslutna enheter till Internet att vara IoT-enheter. Dessa enheter kommer att ansluta tidigare icke anslutna energibegränsade system till internet. Därför är det viktigt att säkerställa säker kommunikation mellan dessa system. Objektsäkerhet är ett koncept där själva datapaketet eller känsliga delar av datapaketet krypteras istället för radiokanalen. Även om en sårbar nod i nätverket kompromisseras så kommer denna mekanism att säkerställa skydd av data. I denna examensrapport föreslås en arkitektur för att använda objektsäkerhetsformatet *COSE* i en typiskt energibegränsad kortdistansradiobaserad IoT plattform. IoT-plattformen använder *Bluetooth Low Energy* och *Constrained Application Protocol* för dataöverföring via en accesspunkt. Implementation baserat på arkitekturen validerar att säkerhetslösningen är genomförbar. En jämförelse mellan nuvarande säkerhetsriktlinjer och den föreslagna objektsäkerhets lösningen ger en liknande overhead-storlek för varje datapaket.

Avhandlingen drar slutsatsen att objektsäkerhet bör ses som ett alternativ för att säkerställa end-to-end säkerhet för ”Internet of Things”.

Abstract

The Internet of Things (IoT) is seen as one of the next Internet revolutions. In a near future the majority of all connected devices to the Internet will be IoT devices. These devices will connect previously offline constrained systems, thus it is essential to ensure end-to-end security for such devices. Object Security is a concept where the actual packet or sensitive parts of the packet are encrypted instead of the radio channel. A compromised node in the network will with this mechanism still have the data encrypted ensuring full end-to-end security. This paper proposes an architecture for using the object security format *COSE* in a typical constrained short-range radio based IoT platform. The IoT platform utilizes *Bluetooth Low Energy* and the *Constrained Application Protocol* for data transmission via a capillary gateway. A proof-of-concept implementation based on the architecture validates that the security solution is implementable. An overhead comparison between current channel security guidelines and the proposed object security solution results in a similar size for each data packet.

The thesis concludes that object security should be seen as an alternative for ensuring end-to-end security for the Internet of Things.

Keywords: *Object security, COSE, IoT security, End-to-end security, CoAP, LWM2M, Bluetooth Low Energy, BLE security*

Acknowledgments

I would like to thank my supervisors Jingcheng Zhang and Patrik Salmela at Ericsson Research. Despite your full schedules you have shown great interest in my work and whenever I was stuck you guided me forward. I would also like to express my gratitude to the entire Ericsson LINLAB team. It has been a great and inspiring year. An extra thanks for all the entertaining table hockey games.

At Linköping university I want to thank Mikael Asplund and Jan-Åke Larsson for their supervision and feedback.

Finally, I wish to thank my girlfriend for giving me strength and motivation.

Linköping, June 2017
Hampus Tjäder

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Problem formulation . . . 2
  - 1.2.1 Current security guidelines . . . 3
  - 1.2.2 Proposed security architecture . . . 3
- 1.3 Objective . . . 4
  - 1.3.1 Related work . . . 4
  - 1.3.2 Research questions . . . 4

2 Theory **5**
- 2.1 Terminology . . . 5
  - 2.1.1 Channel Security . . . 5
  - 2.1.2 Object Security . . . 5
  - 2.1.3 Constrained Systems . . . 6
- 2.2 Bluetooth Low Energy . . . 6
  - 2.2.1 Generic Attribute Profile . . . 7
  - 2.2.2 Advertisement . . . 8
  - 2.2.3 Security modes . . . 8
  - 2.2.4 Security Manager . . . 10
- 2.3 Constrained Application Protocol . . . 11
  - 2.3.1 Message transmission . . . 11
  - 2.3.2 Message format . . . 12
  - 2.3.3 Message operation . . . 13
  - 2.3.4 Datagram Transport Layer Security . . . 13
  - 2.3.5 Lightweight machine-to-machine . . . 14
- 2.4 Authenticated Encryption . . . 14
- 2.5 Concise Binary Object Representation . . . 15
- 2.6 Concise Binary Object Representation with Object Signing and Encryption . . . 16
  - 2.6.1 Object structure . . . 16
  - 2.6.2 Encryption . . . 18
  - 2.6.3 Additional Data . . . 18
- 2.7 Energy consumption . . . 18

3 Security Architecture **21**
3.1 Feasibility Study . . . 21
3.2 System Overview . . . 22
3.3 Proposed COSE object . . . 23
3.3.1 Message structure . . . 23
3.3.2 Message structure expansion . . . 25
3.3.3 Additional Authenticated Data . . . 26

4 Implementation **27**
4.1 System Overview . . . 27
4.2 Sensor . . . 27
4.2.1 Dependencies . . . 28
4.2.2 Bluetooth profile . . . 28
4.2.3 Limitations . . . 29
4.3 Gateway . . . 29
4.3.1 Dependencies . . . 30
4.3.2 Bluetooth profile . . . 30
4.3.3 Lightweight Machine-to-Machine client . . . 30
4.4 Cloud . . . 31
4.4.1 Dependencies . . . 31
4.4.2 Lightweight Machine-to-Machine server . . . 31
4.5 Object Security . . . 31
4.5.1 Sensor support . . . 32
4.5.2 Sender . . . 32
4.5.3 Receiver . . . 32

5 Evaluation **33**
5.1 Testing . . . 33
5.2 Efficiency analysis . . . 36
5.2.1 Bluetooth Low Energy overhead . . . 36
5.2.2 Constrained Application Protocol overhead . . . 37
5.2.3 Overall efficiency . . . 38

6 Discussion **39**
6.1 Research Questions . . . 39
6.1.1 Architecture . . . 39
6.1.2 Implementation . . . 39
6.1.3 Efficiency analysis . . . 40
6.2 Conclusion . . . 41
6.3 Future works . . . 41

A Code examples **45**
A.1 Object Security sender example . . . 45
A.2 Object Security receiver example . . . 46

Bibliography **47**

Notation

Abbreviations

Abbreviation	Description
ADT	Advertising Data Type
AEAD	Authenticated Encryption with AssociatedData
AES	Advanced Encryption Standard
BLE	Bluetooth Low Energy
CBOR	Concise Binary Object Representation
CoAP	Constrained Application Protocol
COSE	CBOR Object Signing and Encryption
DTLS	Datagram Transport Layer Security
GAP	Generic Access Protocol
GATT	Generic Attribute Profile
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IPSO	Internet Protocol Smart Object
IV	Initialization Vector
JSON	JavaScript Object Notation
LWM2M	Lightweight Machine to Machine
M2M	Machine to Machine
MAC	Message Authentication Code
MIC	Message Integrity Check
OMA	Open Mobile Alliance
OSCoAP	Object Security over CoAP
PSK	Pre-Shared Key
PDU	Protocol Data Unit
SDK	Software Development Kit
SICS	Swedish Institute of Computer Science
SRR	Short-range radio
TLS	Transport Layer Security
TCP	Transmission control protocol
UDP	User Datagram Protocol
UUID	Universally unique identifier

1

Introduction

The Internet of Things (IoT) is a term used to describe the increasing number of connected embedded systems. Many of these systems are battery-driven and must therefore have a low power consumption. The IoT can be divided into wide-range and short-range devices. Wide-range IoT are devices that can communicate directly over Internet using cellular telecommunications bands. Short-range IoT devices instead use short range radio wireless technologies which have low transmission power, thus more suitable for devices that require low power consumption. Short-range devices are expected to soon be the majority of all connected devices to the Internet, according Ericsson mobility report from November 2016. By the end of 2016, there were 5.2 billion connected short-range IoT devices. These will increase to 16 billion by 2022 [1]. The short-range IoT devices can in the future be a part of your home, workplace or maybe even your car. Securing the IoT is important from an ethical perspective for each user, since insufficient security could jeopardize personal integrity, system availability and data forgery. IoT devices will affect many areas in society and therefore, it is essential to ensure end-to-end security for such devices.

Since most IoT devices will be short-ranged they will likely use a capillary gateway for communication over Internet. Therefore, an IoT system usually consists of short-range radio (SRR) networks, a capillary gateway, and a web service. The short-ranged radio in this research will be Bluetooth Low Energy (BLE). BLE devices do not natively support IP, thus the capillary gateway must communicate with the sensor over a BLE channel [2]. The gateway will later use the Constrained Application Protocol (CoAP) for communication with the cloud. CoAP is designed for constrained node networks and uses minimal resources, both on the device and on the network [3]. An overview of the IoT platform can be seen in figure 1.1.

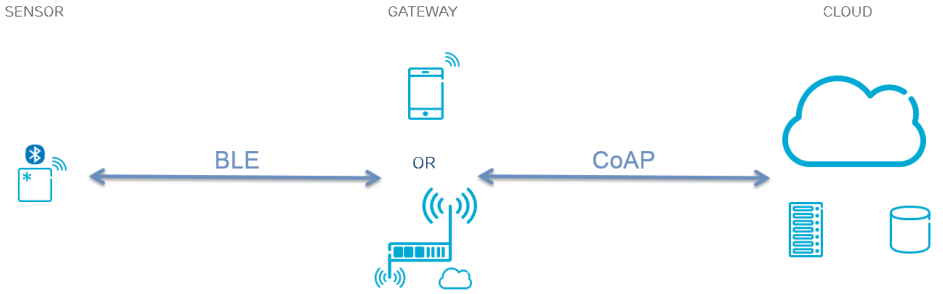

Figure 1.1: *Overview of the described IoT platform.*

1.1 Motivation

The current security guidelines for the described IoT platform do not ensure end-to-end security. The data channels will ensure encryption and authentication but the data is still vulnerable when routing between different protocols. A possible solution to this is object security.

Object security is a concept where the actual packet or sensitive parts of the packet is encrypted instead of the radio channel. The encrypted packet can only be successfully decrypted by the receiving node having the encryption key. By using this mechanism, even if the gateway, or any other node is compromised, the message is still encrypted and cannot be jeopardized. COSE is a proposed standard for object security in constrained networks and will be used in this research [4].

1.2 Problem formulation

Security enhancements for constrained systems are today ensured with traditional channel security. Short-ranged IoT platforms must rely on the security mechanism provided from the SRR technology. Later, the communication between gateway and the cloud must rely on the security mechanism defined by the used protocol. This security architecture does not ensure end-to-end security but instead a hop-to-hop security solution.

The data channels will ensure encryption and authentication but the data is still vulnerable when routing between different protocols. There is also a possibility that a constrained node is selfish and unwilling to participate in the energy consuming security enhancement due to its limited resources. Even if mitigation measures have been made in all nodes in a constrained system, the vulnerability still exists [5].

1.2.1 Current security guidelines

In this research BLE and CoAP will be used as communication channels. According to the Bluetooth core specification, the best practice for ensuring security over BLE is to use LE security mode 1, level 4 [6]. The recommended security mechanism over CoAP is to use the Datagram Transport Layer Security (DTLS) [7]. An overview of the current security guidelines for the described IoT platform can be seen in figure 1.2.

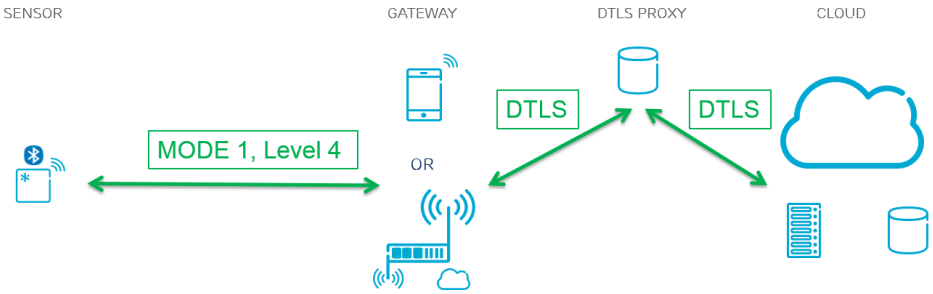

Figure 1.2: *Overview of the current security guidelines in the used IoT platform.*

CoAP specifies the use of proxies for efficiency and scalability. Proxy operations on CoAP messages require DTLS to be terminated at the proxy. The proxy allows adding DTLS encryption without modifying the server code. This architecture is vulnerable, since if the proxy is compromised, an attacker can control the full data flow. This allows modification and creation of new packets, without getting detected by DTLS [8].

The DTLS handshake is also network intensive, even if the least demanding mode based on Pre Shared Key is used. The handshake will also resend many packets on a lossy network [8]. Efficiency evaluation on the performance of DTLS-PSK has shown that the success rate of handshakes drops rapidly after a loss rate with 20 packets on the network, thus retransmission is needed [9]. A security solution based on object security will not have this extra overhead, but may instead have a larger overhead for each packet.

1.2.2 Proposed security architecture

The Concise Binary Object Representation with Object Signing and Encryption (COSE) is a proposed standardized data format for object security [4]. COSE is designed as a lightweight format for object security in constrained node networks. Several research groups drive the standardization process, where Ericsson and the Swedish Institute of Computer Science (SICS) both are contributors [4]. COSE is an extended version of the JavaScript Object Notation (JSON) data model [10].

Using COSE only for ensuring end-to-end security in the described IoT platform comes with some vulnerabilities. The CoAP header is not protected. An

attacker could change the method code from GET to REMOVE and thereby deleting a resource. This should be mitigated. As an additional security preference, the gateway should not have access to any encryption or authentication keys for the COSE object, but integrity should still be assured.

1.3 Objective

The thesis is performed on initiative by Ericsson Research and the aim of the thesis is to explore the possibility of using object security in constrained environments such as Bluetooth Low Energy for ensuring end-to-end security. This will be achieved by first proposing an end-to-end security architecture based on object security and COSE. The proposed security architecture will then be implemented and validated. Therefore, the thesis can be divided into a theoretical research part and an implementation part.

1.3.1 Related work

Object Security over CoAP (OSCoAP) is an IETF working proposal for sending COSE objects over CoAP. The core contributors are Ericsson and the Swedish Institute of Computer Science (SICS) [11]. OSCoAP ensures integrity for the CoAP header and has also proved to be an alternative for DTLS. Efficiency comparison between OSCoAP and DTLS have been performed with the results that OSCoAP in some cases outperforms DTLS [8]. However, COSE objects will in this research not only be transmitted over CoAP but also over BLE. Additionally, the gateway should not have access to authentication keys for the COSE objects and therefore, OSCoAP cannot be used in its current form.

Vucinic et al. have adapted a protocol called OSCAR [12]. The protocol uses a different format than COSE but still ensures end-to-end object security over CoAP.

No related work for using object security over BLE has been found and therefore the results over BLE is of special interest.

1.3.2 Research questions

The research questions are defined as:

- How can object security be used over Bluetooth Low Energy for ensuring end-to-end security?
- Is object security as a solution for end-to-end security implementable in practice for a system operating over different radio networks?
- How efficient in terms of overhead is the proposed solution based on object security compared to channel security?

2

Theory

In this chapter relevant theory for understanding the thesis is provided. Of special interest are current security guidelines and the headers for the used channel protocols. This information will be the cornerstone for being able to propose a security solution based on object security. Details about the COSE format are given in the end of the chapter.

2.1 Terminology

This section will describe some terms that are frequently used in the thesis.

2.1.1 Channel Security

Channel security is a well used term in information security. It provides a secure channel over which objects may be transferred transparently. The channel does not have any special interest about object boundaries and handles the data without any knowledge about the payload. A secure channel is usually managed by a protocol at the transport or network level in the protocol stack [8].

2.1.2 Object Security

Object security is another term used in information security which ensures security without the need of a secure channel. Instead the actual object or sensitive part of the object is encrypted. The encrypted object can only be successfully decrypted by the destination device having the encryption key. Confidentiality and integrity will be provided in the application layer for ensuring secure communication [13]. This is one of the key differences compared to channel security.

2.1.3 Constrained Systems

Constrained nodes and Machine-to-Machine (M2M) are commonly used terms when describing IoT. Both IoT and M2M can be described as networked systems partly or fully based out of constrained nodes. A constrained node can be seen as a constrained device in the system with limited resources compared to a traditional Internet node. A limitation could be the available power and energy, as the case is for battery driven sensors [14]. The network that is used in an IoT system could also be constrained. A network is considered constrained if it has a low throughput or is unreliable [14]. CoAP is a constrained network since it is based on the unreliable nature of UDP [15]. Bluetooth Low Energy could also be seen as a constrained network due to its low throughput [6].

With the nature of constrained systems come new security challenges, which could not be found in traditional Internet platforms [16]. Unreliable networks come with possible integrity vulnerabilities. Networks with low throughput and devices with low power supply are often focused on lifetime, which in turn may affect the used security mechanisms [16].

2.2 Bluetooth Low Energy

Bluetooth Low Energy is a wireless personal area network technology marketed and designed by the Bluetooth Special Interest Group (Bluetooth SIG). The technology is also known as Bluetooth Smart and it was first introduced with the adoption of the Bluetooth Core 4.0 specification. The latest version of the technology is currently Bluetooth 5.0 which was published in December, 2016 [6]. Information in this section is taken from this specification if nothing else is stated.

The Generic Access Profile (GAP) provides a framework for BLE that handles the procedures related to profile roles, discovery of devices, connecting devices, security modes and security. It is the cornerstone that allows BLE devices to interoperate [17]. The relationship of GAP with lower layers of the Bluetooth architecture can be seen in figure 2.1. In this thesis there is mainly one module of GAP that is of interest, which is the Generic Attribute Profile (GATT). The rest of the lower layers will not be described since they are irrelevant for the results.

Figure 2.1: *Overview of GAP in the Bluetooth architecture.*

2.2.1 Generic Attribute Profile

The Generic Attribute Profile (GATT) establishes how to exchange all profile and user data over a BLE connection. GATT uses the Attribute Protocol (ATT) as transport protocol. The data hierarchy in a GATT profile is grouped into services, which contain zero or more characteristics. A characteristic can be described as containers for user data consisting of various attributes. The *value* attribute holds the actual user data. Operations and procedures that can be used by the characteristic is described in the *properties* attribute. One property could for example be that the user data in the characteristic can only be read and not overwritten. The *descriptor* attribute can be used to provide additional information about the characteristic. An overview of the GATT data hierarchy can be seen in figure 2.2.

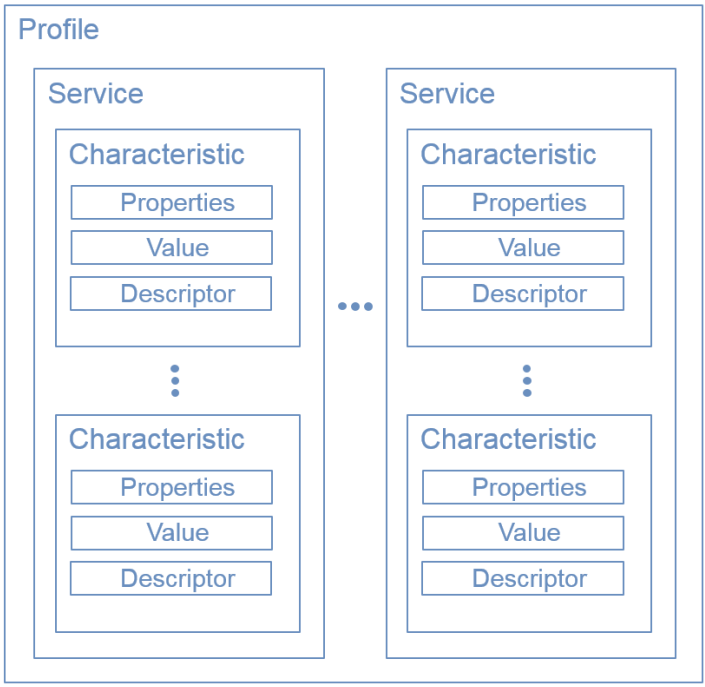

Figure 2.2: *Overview of the GATT profile architecture.*

A universally unique identifier (UUID) is a 128-bit number guaranteed to be unique [18]. UUID's are used in GATT profiles for identifying services and characteristics. If multiple devices should have support for the same service, then the same UUID should be used for all devices. Hence, the identifier will tell what kind of data that the BLE device uses. The BLE specification adds two additional UUID formats for efficiency possibilities. These have a size of 16-bits respectively 32-bits. To reconstruct a full 128-bits UUID from a shortened version the Bluetooth Base UUID is used, where the first bytes represent the short value:

- xxxxxxxx-0000-1000-8000-00805F9B34FB

2.2.2 Advertisement

Bluetooth Low Energy is based upon advertisement of data. The format which the data is carried in when advertising is defined in the GAP specification. It consists of a sequence of data structures, each starting with one byte defining an Advertising Data Type (ADT), followed by the actual data of variable length.

Roles

GAP specifies four different advertisement roles that can be adopted by a device in a BLE network [17]:

- Broadcaster: Designed for transmit-only applications that will periodically send out advertising packets of data.
- Observer: Designed for receive-only applications that want to collect data from broadcasting devices.
- Central: Device will act as Link Layer master. Capable of establishing multiple connections and is always the one that initiates a connection.
- Peripheral: Device will act as Link Layer slave. Allows central to find the device using advertising packets.

2.2.3 Security modes

The security requirements of a BLE device are expressed in terms of a security mode and security level. There are two LE security modes. A connection between two devices shall operate in only one security mode.

Security mode 1

The first mode consists of 4 different levels where authentication is achieved by enabling encryption. The first level provides no security and is the initial state for each connection. A connection starts its lifetime in this state and can later be upgraded to any mode and level. The highest security available for a BLE connection uses the first mode with security level 4. This is the latest security update for the BLE core and utilizes stronger pairing procedures, which in turn gives better encryption. It was first introduced in Bluetooth 4.2 as LE secure connections and is the recommended security solution to use.

The Link Layer provides authentication and encryption using Counter with Cipher Block-Message Authentication Code (CCM) Mode, also known as AES-CCM [19]. In the BLE specification the Message Authentication Code is called Message Integrity Check (MIC) and to avoid confusion this will be the term used here as well. CCM has two size parameters defined in the Link Layer which are 4-bytes MIC and a 2-bytes Length field.

The MIC is calculated with the counter-mode blocks over the data channels payload field and the first byte of the header. A field for the MIC is then appended to the payload, which ensures authentication. Encryption is then applied

to the data channel payload and the MIC. According to the specification of CCM a unique nonce value is required for each Data channel PDU. This nonce is 13-bytes large in BLE, of which 7-bytes are used as Initialization Vector (IV) and 4-bytes are used as sequence number [6]. The sequence number is an incremented value starting from zero and the IV is a unique randomized vector. The nonce will be a part of a 16-bytes encryption block, which is used to encrypt and decrypt the payload. The encryption block will not be sent for each data packet, but instead the IV is exchanged during the handshake procedure. Parts of the payload are later used for *XOR*ing the IV, thus ensuring a unique nonce for each packet. Therefore, the 4-bytes MIC is the only overhead added to each sent BLE data packet.

Second mode

In the second mode, data signing is used for transferring authenticated data between two devices in an unencrypted connection. The method can be used by services that require fast connection set up and fast data transfer. For each data packet a signature is added for ensuring integrity. The signature consists of a SignCounter and a Message Authentication Code (MAC) generated by the signing algorithm. The MAC is generated using the shared secret key and the data to authenticate, thus it will be unique for each message. The shared secret is in BLE known as the Connection Signature Resolving Key (CSRK) and it is distributed during the pairing process. The SignCounter is an incremented integer for protection against replay attacks [6]. An overview of the mode can be seen in figure 2.3.

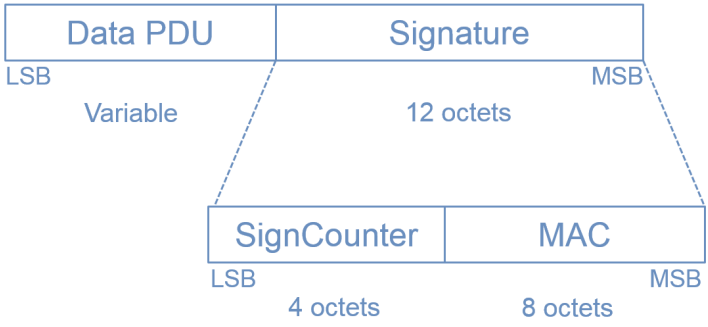

Figure 2.3: *Overview of LE secure mode 2 architecture.*

2.2.4 Security Manager

The Security Manager can be found inside of the Generic Access Profile. It defines methods of pairing and key distribution, a protocol for those methods and a cryptographic toolbox to be used. In relation to the LE architecture it can be seen as a layer between GAP and the link controller [6].

Pairing

Pairing is performed to establish keys which can then be used to encrypt a link. The security of the encryption is impacted by the type of pairing performed. Pairing is a three-phase process. The first two phases are always used and they may be followed by an optional transport key distribution.

1. In the first phase, the devices exchange authentication requirements and IO capabilities to determine which pairing methods shall be used in the second phase. Authentication requirements retrieved from the Pairing Feature Exchange also determine whether LE legacy pairing or LE Secure Connections is used. LE Secure Connections was first introduced in Bluetooth 4.2 and is mostly an improved version of LE legacy pairing. LE Secure connections utilizes Elliptical Curve Hellman-Diffie (ECDH) as key exchange procedure which ensures better security than the previously unique key exchange procedure for BLE.

2. In the second phase the selected pairing method is performed.

3. In the third phase, the key distribution is performed for sharing required keys, which can be used to encrypt a link in future reconnections or to sign data.

Cryptography

Inside the security manager a cryptographic toolbox is provided that handles generation of keys and other cryptographic algorithms. Encryption in Bluetooth low energy uses AES-CCM cryptography with a 128-bit key. AES encryption is considered to be very secure, but the key exchange protocols that BLE uses introduce severe security vulnerabilities which may allow an attacker to decrypt the data. Therefore, the pairing method used for exchanging the keys has great effect on the security of the connection.

2.3 Constrained Application Protocol

The Constrained Application Protocol (CoAP) is an IETF standard designed for asynchronous web transfer over constrained networks. It is a popular REST [20] protocol used for communication in IoT platforms. CoAP is a lightweight version of the HTTP protocol and therefore CoAP integrates well with existing web services [3]. Efficient is an important requirement due to the lightweight nature of the CoAP protocol. That is why the protocol is based upon UDP instead of TCP [21]. An overview of the CoAP stack utilizing OMA LWM2M can be seen in figure 2.4.

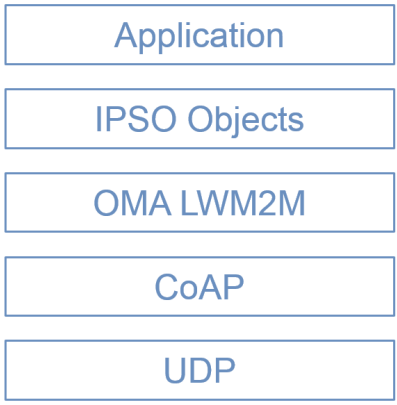

Figure 2.4: *Overview of the CoAP stack in the used IoT platform.*

Layers of the used CoAP stack will be described next in this section. The reader is expected to have knowledge of UDP, hence the description will start from the CoAP layer. LWM2M and IPSO will later be described in section 2.3.5.

2.3.1 Message transmission

CoAP uses an asynchronous messaging model. As a result, CoAP is designed to work primarily through proxies, which ensures efficiency [3] [11].

The request methods supported in CoAP are DELETE, GET, POST and PUT. These can easily be mapped to HTTP and thus have the same properties as in HTTP. The type of a CoAP response message can be identified by the response code field in the header [22]. There are four different response types supported in CoAP which are [3]:

- Confirmable: Messages that requires an acknowledgement packet.
- Non-Confirmable: Messages that do not require an acknowledgement.

- Acknowledgement: Response message that acknowledges the receipt of a confirmable message identified by the Message ID.
- Reset: Sent in response to a confirmable or non-confirmable indicating that the message cannot be processed.

2.3.2 Message format

CoAP messages are encoded after a binary format divided into header and payload. The message starts with the header which is of fixed 4 bytes size. The header may be followed by a Token value, CoAP options and optionally a payload. [4] The message structure can be seen in figure 2.5.

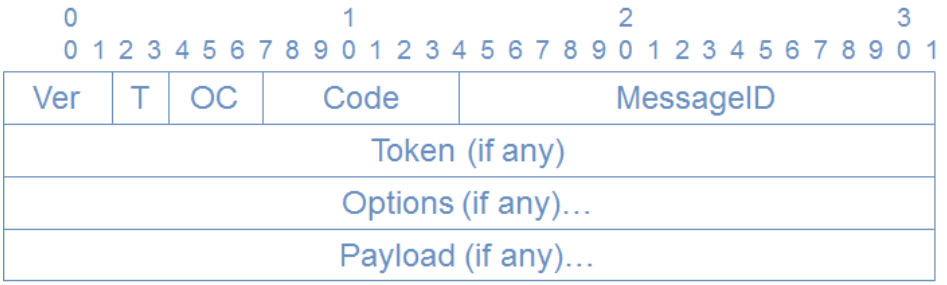

Figure 2.5: *Overview of the CoAP message structure.*

The fields in the CoAP message are defined as:

- Version (Ver): 2-bit unsigned integer which indicates the CoAP version.
- Type (T): 2-bit unsigned integer which indicates whether a message is confirmable, non-confirmable or a confirmation.
- Token Length (TKL): 4-bit unsigned integer which indicates the length of the Token field.
- Code: 8-bit unsigned integer which describes Request/Response code. Corresponding to HTTP codes, e.g. POST or GET.
- Message ID: 16-bit unsigned integer used to detect message duplication and acknowledgement handling.
- Token: 0-8 bytes value which is used to correlate responses and requests.
- Options: There are a number of options that can be added to the message. An example could be how the receiver should handle the message and with what resources. Options are explained more in details below.
- Payload: Contains any payload of a message.

2.3.3 Message operation

Operations are an essential part of CoAP. The header is short and almost all functionality over CoAP is performed with options. There are several options that can be included in a CoAP message. The functionality of CoAP options is versatile and it is easy to add more options to the protocol. An option could for example be to set the time that a message is valid. Each option specifies the number of the defined CoAP option, the length of the option value, and the value itself. The format of the option should follow the same order. Options are transported in an ordered list as a part of the CoAP header. [3]

2.3.4 Datagram Transport Layer Security

The Datagram Transport Layer Security (DTLS) is an extended version of the Transport Layer Security (TLS) [23]. TLS is an IETF security standard for the transport layer and is widely used for ensuring security over HTTP. A website using TLS is commonly said to use HTTPS. TLS is however designed for operating over TCP and is therefore not suited for the UDP nature of CoAP [21]. DTLS is an adapted IETF security standard for use with UDP instead of TCP [7]. It is the recommended security guideline to use DTLS over CoAP [24]. Since DTLS is based on TLS it also inherits the use of authenticated encryption and AES-CCM for ensuring security. A commonly used mode in CoAP and DTLS is to utilize Pre-shared Key (PSK) distribution mode, thus the 128-bit encryption key is explicity known by both nodes [25].

Packet overhead

The DTLS record layer is an encapsulating protocol used to transport data. It has a size of 13-bytes and consists of:

- 1-byte content type field which states the protocol used to process the enclosed fragment.
- 2-bytes DTLS version field for recipient and proxy handling.
- 2-bytes epoch field, which is a counter value that is incremented for every cipher state change.
- 6-bytes sequence number for protection against replay attacks.
- 2-bytes length field for message size.

The record layer is not the only packet overhead and with the use of AES-CCM a 8-bytes Integrity Check Value (ICV) is added for each packet. In addition to the ICV, a 8-bytes nonce is added for ensuing a unique encryption. This nonce can be described as an Initialization Vector (IV). Therefore, the overhead for a packet sent over DTLS has 13-bytes record layer, 8-bytes ICV and 8-bytes IV, thus resulting in total 29-bytes. [26]

Handshake

The unreliable nature of UDP requires DTLS to have a more expensive handshake procedure compared to TLS. An expensive handshake is required for setting up the link layer encryption, even if DTLS with PSK mode is used [25]. The handshake procedure over DTLS-PSK consists of 6 different transmission phases, which together require hundreds of bytes being transmitted between the nodes [26].

2.3.5 Lightweight machine-to-machine

OMA Lightweight M2M is a protocol for M2M and IoT device management. It is standardized by the Open Mobile Alliance, which develops open standards for the mobile phone industry [27]. The protocol is designed for data transmission in constrained environments and it uses CoAP for data exchange. The aim with the standard is to develop a fast deployable client-server specification to provide M2M service for the emerging IoT systems. With an object based resource model architecture it is possible to utilize IPSO standardized objects for data transmission [28]. The IPSO Alliance focuses on enabling IoT devices to communicate based on open standards. Leshan is an open-source OMA LWM2M client and server library published by Eclipse [29]. Based in Java, the library can easily be accessed by using Maven.

By ensuring security over CoAP the protocol will be secure as well [30].

2.4 Authenticated Encryption

Both DTLS and BLE utilize the same cryptographic concept for ensuring security. High-level details about the procedure will be presented in this section. Authenticated encryption is when the encryption simultaneously provides integrity, authenticity and confidentiality protection of the data. AES-CCM is a mode of operation for cryptographic block ciphers, which is very secure [19]. The algorithm is used to ensure security over several protocols, such as TLS, IPSEC and BLE to mentioning a few [23]. CCM stands for Counter with CBC-MAC, thus a combination of CBC-MAC and the Counter (CTR) mode [31]. This construction gives the possibility to choose which parts that need both integrity and confidentiality and which parts that should only be integrity protected. This makes it an algorithm for Authenticated Encryption with Associated Data (AEAD) [32]. Authenticated encryption and decryption are the two supported operations in the algorithm.

The inputs for the authenticated encryption can be seen in figure 2.6. The output is the ciphertext which may incorporate an authentication tag [33].

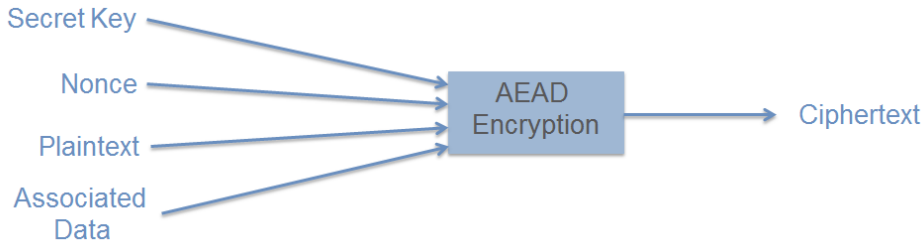

Figure 2.6: *AEAD encryption*

The inputs for the authenticated decryption can be seen in figure 2.7. The output is either the plaintext or a failure indicator. [33]

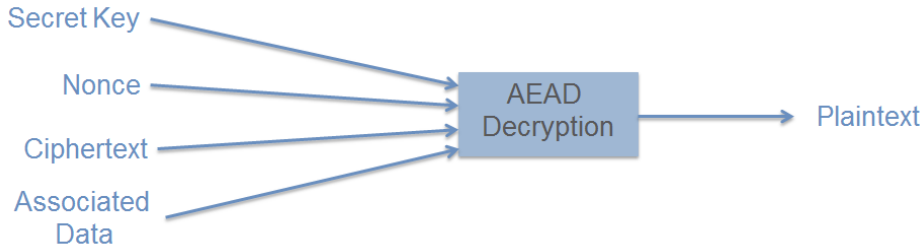

Figure 2.7: *AEAD decryption*

2.5 Concise Binary Object Representation

The Concise Binary Object Representation (CBOR) is an IETF data format standard designed for constrained node networks. The primary focus lies on very small code and message size, which are desired qualities for a constrained environment. Compared to other binary representation formats the underlying data model for CBOR is an extended version of the JSON data model [10]. The data structure is therefore based on a key-value object representation. Before transmission, the data is encoded and compressed down to a binary sequence. In the receiving node, the binary sequence is decoded into a readable format. This data compression ensures usage of minimum bytes for sending a packet. [34]

2.6 Concise Binary Object Representation with Object Signing and Encryption

CBOR with Object Signing and Encryption (COSE) is a working proposal standard data format for object security. COSE was designed with the purpose to have a lightweight format for object security in constrained node networks. It provides the possibility to create and process signatures, message authentication codes and encryption using CBOR for serialization. [4]

2.6.1 Object structure

The object structure in COSE is designed for using a large amount of common code when processing and parsing different types of security messages. All message structures are CBOR array types, which always have the first three elements:

- Protected: Contains parameters in the current layer that should be encrypted.
- Unprotected: Contains parameters about the current layer in plain text.
- Content: The content is either the plain text or the cipher text.

Depending on the message type, there can also be additional elements. Messages can be separated into layers based on different types of cryptographic concepts. Identification of a message structure can be provided by adding a CBOR tag at the start of a COSE message. There are three different message structures targeting a single recipient which can be seen in table 2.1. These structures will be described in detail below the figure.

CBOR Tag	Message structure	Semantics
16	Encrypt0	Encrypted data object
17	Mac0	Mac data object
18	Sign1	Signed data object

Table 2.1: *COSE message structures for a single recipient*

Encrypt data object

The encrypted data object in COSE utilizes authenticated encryption and AES for ensuring security, as described in section 2.4. An initialization vector is therefore included in the object for ensuring uniqueness of the encryption [35]. It is possible to extend the message structure with additional parameters such as a signature, used algorithm and partial IV etc.

An example of how the Encrypt message structure may look like can be seen in figure 2.8.

```
16(
  [
    / protected / h'a1010a' / {
        \ alg \ 1:10 \ AES-CCM-16-64-128 \
      } / ,
    / unprotected / {
      / iv / 5:h'89f52f65a1c580933b5261a78c'
    },
    / ciphertext / h'5974e1b99a3a4cc09a659aa2e9e7fff161d38ce7edd5617
388e77baf'
  ]
)
```

Figure 2.8: *Encrypt message structure example from the COSE specification.*

Signed data object

The signed data object in COSE utilizes Elliptic Curve Digital Signature Algorithm (ECDSA) for ensuring integrity protection [36]. The smallest signature size in described the COSE specification is 64-bytes. A signature can be added to the Encrypt1 message structure, thus it is possible to combine the two structures for ensuring AES encryption with ECDSA signature protection [4].

An example of how the Sign1 message structure may look like can be seen in figure 2.8.

```
18(
  [
    / protected / h'a10126' / {
        \ alg \ 1:-7 \ ECDSA 256 \
      } / ,
    / unprotected / {
      / kid / 4:'11'
    },
    / payload / 'This is the content.',
    / signature / h'eae868ecc176883766c5dc5ba5b8dca25dab3c2e56a551ce
5705b793914348e19f43d6c6ba654472da301b645b293c9ba939295b97c4bdb84778
2bff384c5794'
  ]
)
```

Figure 2.9: *Sign1 message structure example from the COSE specification.*

Mac data object

The mac data object in COSE utilizes CBC-MAC for ensuring integrity protection. A truncated message authentication code is added to the object which ensures protection from forgery [37].

An example of how the Mac0 message structure may look like can be seen in figure 2.8.

```
17(
  [
    / protected / h'a1010f' / {
        \ alg \ 1:15 \ AES-CBC-MAC-256//64 \
      } / ,
    / unprotected / {},
    / payload / 'This is the content.',
    / tag / h'726043745027214f'
  ]
)
```

Figure 2.10: *Mac0 message structure example from the COSE specification.*

2.6.2 Encryption

COSE supports many different cryptographic concepts. The encryption will be based on a pre-shared (PSK) key distribution, where the encryption key is given to the sensor during manufacturing. COSE has support for key exchange mechanisms but these will not be used due to scope of the thesis. Authenticated encryption will be provided with AES-CCM [38]. AES is the cryptographic primitive and CCM is a mode of operation for cryptographic block ciphers which provides both integrity and confidentiality protection.

2.6.3 Additional Data

COSE comes with a feature for ensuring authentication for externally supplied data. This data will not be a part of the COSE object itself and therefore never transmitted. The primary reason for this feature is to support integrity protection of, for example header attributes. By doing so it is possible to protect sensitive data and protocol headers from forgery [4].

2.7 Energy consumption

Efficiency research comparing application-layer based security and link-layer based security has shown that application based security tend to consume more energy when performing cryptographic calculations. COSE also uses an extra parse and serialization step since CBOR is used. A recent efficiency analysis of COSE with the OSCoAP format demonstrated that indeed, COSE was slightly more energy consuming in the end-nodes compared to DTLS. The efficiency analysis was performed by two students doing their master thesis's [8]. The CPU time for cryptographic operations given by the studied paper can be seen in table 2.2. Cryptographic support was provided by mbedTLS.

In the tested system from the studied paper the decryption had 0.3 ms longer CPU time for each packet. The encryption of a packet took 0.45 ms longer CPU time for each packet. This is not a large increased but as a result the energy consumption will be slightly higher in the end nodes. The increased energy consumption for cryptographic functions can also be verified in another similar analysis over OSCoAP. However, the author said that the cryptographic implementation at the testing point was unoptimized which affected the result [33].

	Decrypt	Encrypt
OSCoAP	1.123 ms	1.327 ms
DTLS	0.8102 ms	0.8717 ms

Table 2.2: *Average CPU time for a packet in cryptographic operations [8]*

Worth mentioning is that the given results from the studied papers had an increased RAM and ROM usage due to a large amount of unused source code [8].

3

Security Architecture

Using COSE as end-to-end security solution in the described IoT platform, seen in figure 1.1, has some vulnerabilities as mentioned in the problem formulation in section 1.2. Therefore, a security architecture using COSE as end-to-end security solution is proposed. In this chapter a feasibility study is first described, which was performed in order to propose a reliable architecture and to be able to implement the actual IoT platform. An overview of the proposed system and how to ensure end-to-end security is then presented. The proposed COSE message architecture which came as a result of the feasibility study is described and motivated in the end of the chapter.

3.1 Feasibility Study

The feasibility study focused on the COSE specification and how the protocol has been used in other systems for ensuring security. COSE is in an early state, most of the studies and publications were conducted as working IETF standardization Internet drafts, in particular the working proposal OSCOAP [11]. This standard claims to protect the CoAP header, which could be used as guideline to also protect the CoAP header in the proposed solution. Since most studies in COSE are drafts, and in order to not miss any important concept with object security, an additional object security format was investigated. OSCAR is another object security solution which uses a different format than COSE [12]. By studying OSCAR it was possible to understand other use-cases of object security, which gave inspiration for enhancement of the proposed COSE message structure in specific cases. Since COSE is used for ensuring security over BLE and CoAP it was essential to include research about these protocol in the feasibility study, since these are the channels that should be protected.

The proposed architecture was also required to provide the same security strength as in the traditional solutions described in the specifications, therefore the current security guidelines were studied in detail. This made it possible to propose an architecture based on object security with the same security strength. The study concluded that AES-CCM was the used cryptographic solution over both BLE and CoAP. For ensuring similar security strength and performing a reliable overhead comparison, AES-CCM was also the used cryptographic algorithm in the proposed object security solution. With an increased knowledge about the object security format and the used system, it was possible to propose an architecture based on COSE for ensuring end-to-end security.

3.2 System Overview

An overview of the proposed end-to-end security enhancement using object security and COSE can be seen in figure 3.1. The architecture assumes that a pre-shared key is used, thus no key exchange will be performed.

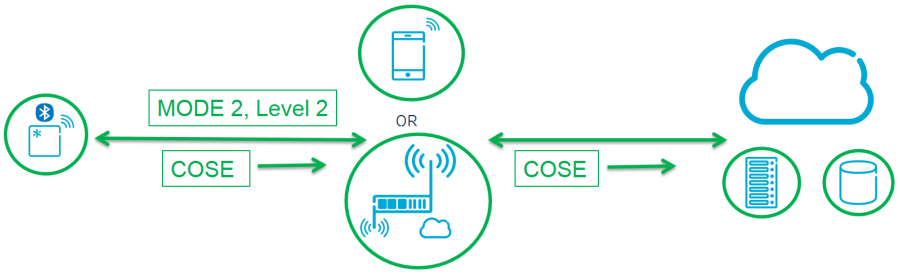

Figure 3.1: *Proposed security architecture using object security*

COSE cannot ensure integrity protection over the BLE link due to the requirement that no object related keys should be known by the gateway. Therefore, integrity protection with access control over the BLE link is required and thus security mode 2, known as data signing, is used. Clearly, this requirement will add extra overhead for the transmission over the BLE link but it will mitigate the vulnerability which comes with key handling in the gateway. The architecture has a vulnerability since the header to the channel in receiving node is not protected by default. This header can be protected using additional authenticated data as described later in section 3.3.3. If COSE is utilized per the standard, then the proposed architecture will ensure end-to-end security, thus if any node is compromised the message is still secure and cannot be jeopardized.

3.3 Proposed COSE object

Depending on use-case the COSE message structure may vary. As mentioned earlier in section 2.6.1, there are three different COSE message structures for a single recipient. These structures are "Encrypt0", "Sign1" and "Mac0". Each message structure ensures various security strengths since different cryptographic algorithms are used. In this research the focus will be at the "Encrypt0" structure and authenticated encryption. The motivation behind this choice was that an easier and more reliable overhead comparison against traditional channel security could be made since the same security mechanism is used there. The COSE specification also proposes this message structure if encryption will be used.

3.3.1 Message structure

Using the "Encrypt0" message structure provides the same security strength with the same algorithms as in both DTLS and LE secure connections mode, namely AES-CCM with a 128-bit key as described in the theory chapter. Both OSCoAP and OSCAR also utilize AES-CCM encryption for ensuring confidentiality [11] [12].

The COSE overhead fields required for ensuring equivalent security were obtained by studying the DTLS and LE security mechanism [6] [7]. Guidelines for how to use AES-CCM with COSE can also be found in COSE specification [4]. Thus, it was possible to propose a message structure that ensures equivalent security strength. The proposed COSE message structure can be seen in figure 3.2. As Initialization Vector (IV) a 7-bytes nonce is used for ensuring uniqueness of the encryption, thus same size as specified in the COSE specification [4]. This vector will vary for each packet and will be randomly generated when creating the object in the sender node. As partial IV a sequence number is set for ensuring protection against replay attacks [6] [11] [12]. In this architecture the sequence number has a size of 1-byte, thus only 256 message can be sent in a connection. Compared to DTLS which has 4-bytes intended for the sequence number it may seem little, but it is enough for the data transmission in our use case [26]. The size can be increased if a larger message support is required.

The architecture gives a message overhead of **25-bytes** for each data packet. This size applies for all different channels a packet is transmitted via, and regardless of payload size. Optionally a message could also include:

- Message tag: for the possibility to handle different message structures in the cloud. This tag has the size of one byte and would indicate in the beginning of the message whether the structure would follow "Encrypt0", "Sign1" or "Mac0" encoding. [4]
- Algorithm field: specification of which cryptographic algorithm that the receiver should use to decrypt the message. This would enable generic cryptographic support for the recipient but add 2 more bytes to the message structure. An assumption in our use case is that the algorithm will

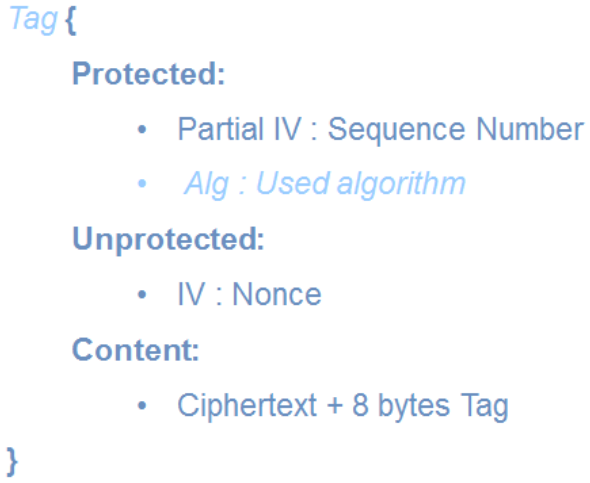

Figure 3.2: *Proposed COSE message structure where italic fields are optional*

be explicitly known by the server and thus not part of the proposed COSE architecture.

When the recipent knows the inititalization vector the sequence number can be used for xoring the vector, ensuring a unique nonce for each packet. This principle is used in both OSCoAP and BLE [11] [6]. For ensuring uniqueness, it is important that the sequence number is unique and checked upon each transmission. When the IV is known by the recipent it is no longer required to send the IV for each packet. Therefore the COSE message structure does not need to include the IV parameter after an initial object is sent. This gives a smaller packet size, which results in 16 bytes overhead for a mandatory packet. The COSE packet sent after the initial will therefore have the the message structure that can be seen in figure 3.3.

Tag {

Protected:

- Partial IV : Sequence Number
- *Alg : Used algorithm*

Content:

- Ciphertext + 8 bytes Tag

}

Figure 3.3: *Proposed shortened message where italic fields are optional*

3.3.2 Message structure expansion

OSCAR utilizes ECDSA signature for ensuring integrity protection of the object [12]. In the OSCoAP specification there are also examples for adding a ECDSA signature to an "Encrypt0" message [11]. It is therefore possible to add a ECDSA signature to the proposed COSE architecture. The smallest size of such a signature in COSE has the size of 64 bytes, which would increase the overhead considerably. However, a new use-case for the cloud provider is possible if this is utilized. The cloud provider then has the possibility to handle other third parties sensitive sensor data without reading it. The signature key will be known by the cloud provider, but not the AES-CCM key. Integrity protection will be validated by the signature and the COSE object will then be stored in the cloud. The third party and the owner of the AES-CCM key could then request the encrypted COSE object from the cloud and decrypt it. Since AES-CCM is used, the transmission from the cloud to the third party will be integrity protected and confidentiality will be assured. The use-case can be visualized in figure 3.4.

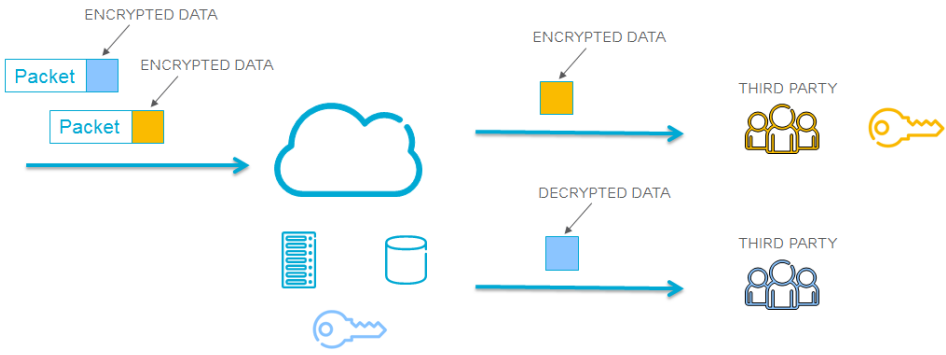

Figure 3.4: *Use-case with and without an additional signature.*

Clearly, by using application layer based security such as COSE it is easy to change the security architecture with minimal effort. The use-case with an additional signature will not be implemented nor analyzed, but it shows the flexibility for using application based end-to-end security solution.

3.3.3 Additional Authenticated Data

The COSE specification gives the possibility to ensure integrity protection of additional data that is not a part of the object itself, as mentioned in the theory chapter. By doing so, it is possible to protect channel headers such as the CoAP header in this case. OSCoAP utilizes this functionality [11]. However, OSCAR does not provide integrity protection of the CoAP header, but the report states that required fields could be included in the key derivation procedure, which is the method utilized in COSE [12] [4].

There are several CoAP header fields that must be read and updated during message exchange or when traversing a proxy. According to the OSCoAP specification there are two header fields of certain interest that can and should be protected. These are the CoAP version number and the CoAP header field code. Any other CoAP header fields shall not be integrity protected since they may change [11]. Protection of the CoAP header is ensured by adding the additional authenticated data to the generation of the AES-CCM encryption. In the receiver, the actual CoAP header fields that were received are added as parameters for the decryption of the message, thus a successful decryption with the pre-shared 128-bit encryption key then verifies integrity. CoAP requests need to be mapped to BLE requests so that the correct header will be protected in advance when encrypting the object. For example, when the sensor receives a BLE read request, this can easily be mapped to a CoAP GET.

4

Implementation

In this section details regarding the implementation of the IoT platform and the proposed security enhancement are presented. An overview of the IoT platform and the used hardware is described. This will be followed up by high-level details about how each node of the system was implemented. Finally, the COSE implementation is presented.

4.1 System Overview

In order to validate the proposed security architecture, an IoT platform consisting of a BLE sensor image, capillary gateway, and a cloud service was required. The testbed for the IoT platform was based on a Nordic NRF52 sensor and a Raspberry PI 3 acting as a capillary gateway. As server, a local computer was used but the system also had a possibility to communicate over a 3GPP USB dongle up to a cloud. The system was continuously tested during the implementation phase.

4.2 Sensor

Implementation of the used Nordic NRF52 sensor will first be described. Dependencies of the implementation will be highlighted. Thereafter, any found limitations in the soft device and the sensor SDK will be presented. Implementation of the sensor image will be described in the last section. The sensor device can be seen in figure 4.1.

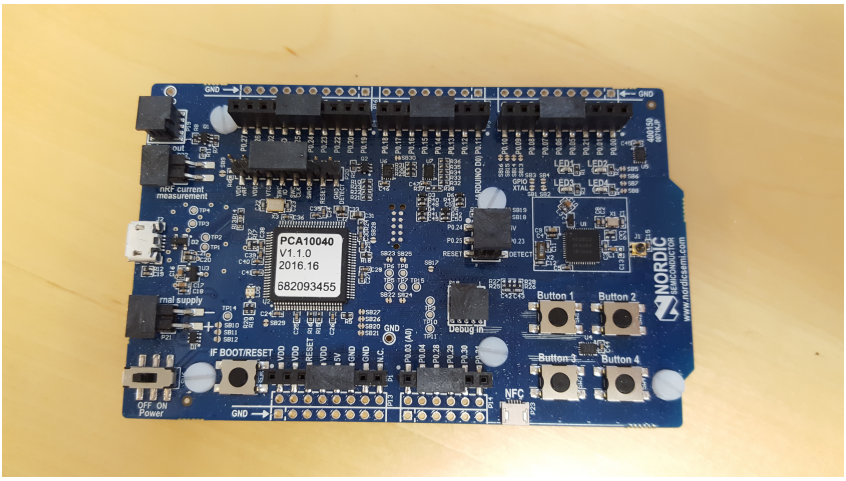

Figure 4.1: *Nordic nrf52 sensor*

4.2.1 Dependencies

The implementation used some specific libraries and tools. These dependencies are listed in this section as:

- Bluetooth Developer Studio 1.1: was used for generating the BLE GATT profile.
- Keil uVision5: was used as IDE
- GNU make: was used for compiling the hex file.
- nRFgo Studio: was used for flashing the hex file to the sensor.
- uART: was used for debugging the source code.
- COSE-C: used COSE library.
- nrf52 SDK 13.0-alpha: version of the sensor's software development kit.
- Softdevice s132 v5.0.0-alpha: version of the sensor's soft device.

4.2.2 Bluetooth profile

Bluetooth Developer Studio was used to generate a custom GATT profile. The profile had one writable characteristic and one readable characteristic for enabling two-way communication. With an available plug-in for the used Nordic sensor some auto-generated code could be provided, which facilitated the integration of the GATT profile with the sensor image. The data size of the used characteristics was increased to maximum with the help of Texas sniffer for finding the

limit when packets no longer was advertised. The advertisement format was then changed for enabling filtering possibilities. The sensor was implemented with a peripheral advertisement role as described in the theory chapter.

GNU make was used for compiling the source code and nRFgo Studio was then used for flashing the generated hex file to the sensor. LEDs on the board was in an early phase used for debugging, but this became complex as the sensor image increased in size, therefore it was later replaced with debugging over uART.

4.2.3 Limitations

Early tests showed some limitations in the sensor SDK by Nordic. The first known limitation was the data size of a single GATT characteristic. Testing confirmed that 255 bytes was the maximum allowed byte length of a characteristic. Thus, the maximum data packet size could be at most 255 bytes in the implementation.

The support of LE security mode 2 in the Nordic sensor was questioned in an early phase of the thesis. Assurance from Nordic in their forums confirmed that the mode was supported. Implementation attempts of the security mode were made but without any success. Discussions with Nordic employees resulted with them admitting that data signing was not yet fully supported by any of their soft devices. Data signing could therefore not be a part of the proof-of-concept implementation due to the lack of support in the sensor's softdevice.

4.3 Gateway

As gateway a Raspberry PI 3 was used. The hardware can be seen in figure 4.2. Dependencies of the implementation are first highlighted and thereafter the forwarding between the two communication protocols is described in a high-level.

Figure 4.2: *Raspberry pi 3*

4.3.1 Dependencies

The gateway implementation used some specific libraries and tools. These dependencies are listed in this section:

- Eclipse Luna: was used as IDE
- Apache Log4j Logger: was used for debugging purposes.
- Raspberry PI image: Linux Debian 8
- Bluez 4.101: The implemented BLE framework was dependent on Bluez callbacks, thus callbacks may vary depending on version.
- Bash: was used for remote communication with the Raspberry PI.
- Maven: was used for library dependencies in the source code.
- Leshan: was used as LWM2M client.
- COSE-Java: used COSE library.

4.3.2 Bluetooth profile

BLE support was achieved by using a self-implemented BLE framework. The framework was implemented with a similar design as the open source framework Eclipse Kura [39], which relies on callbacks from Bluez operations. By looking after BLE advertisement with a specific format it was possible to filter out irrelevant beacons. The gateway was implemented with a central advertisement role. When a device comes in proximity of the gateway a connection attempt is made. If successful, a BLE service discovery is performed to investigate supported GATT services on the device. This procedure allows a generic creation of supported IPSO smart objects later used in the LWM2M setup.

4.3.3 Lightweight Machine-to-Machine client

After the service discovery phase was finalized the gateway starts a LWM2M client specific for the connected device, thus the gateway forward messages between the sensor and the cloud via this client. The data exchange uses the IP smart objects format. After a successful registration with the LWM2M server is made the system is fully operative for asynchronous two-way communication end-to-end.

4.4 Cloud

The server acting as a cloud is based on CoAP and the LWM2M protocol Leshan. The dependencies of the server side are first highlighted, where support of external libraries were given by using Maven. At last, the implementation and usage of Leshan is briefly described at a high level.

4.4.1 Dependencies

The implementation of the cloud used some specific libraries and tools. These dependencies are listed in this section:

- Eclipse Luna: was used as IDE.
- Maven: was used for library dependencies in the source code.
- Apache Log4j Logger: was used for debugging purposes.
- Leshan: was used as LWM2M server.
- Windows Laptop: was used as server platform.
- COSE-Java: used COSE library.

4.4.2 Lightweight Machine-to-Machine server

When a connection with a new LWM2M client is established the registration phase is started. Upon successful registration a unique ID is generated for the channel which is used for further communication. Also, when a new client is registered an observation is added to the LWM2M node resource used for CoAP notification sent from the sensor. This observation allows us to asynchronously receive data packets from the sensor at any time. Thus, the communication then supports two-way communication with the sensor. With a listener added to the observed resource it is easy to map each notification to the COSE implementation for verification of the packet. Important parts of the CoAP header are sent as parameters into the COSE object handler class for allowing generation of additional authenticated data.

4.5 Object Security

In this section the implementation of the proposed COSE architecture is presented. The library worked flawlessly over the gateway and the server which was based on the Java version of the library. However, some modification of the C-based source code was needed to add support in the sensor image. These modifications will first be described. With support in all nodes, the actual proof-of-concept implementation of the proposed COSE object architecture is presented in the last sections. The implementation will verify the flow of a CoAP GET request, with the response of a CoAP notification containing the generated COSE object.

4.5.1 Sensor support

One of the biggest implementation challenges was to modify the COSE library for support of the Nordic nrf52 sensor. COSE was originally made for operating over IP networks, thus some implementation needed to be removed from the library since the sensor did not support IP but only native BLE communication. Also as default, COSE utilizes OpenSSL as cryptographic library which was not possible to use on the embedded device. Therefore, mbedTLS was manually added to the sensor image which gave sufficient support for required cryptographic algorithms. An additional issue with the COSE implementation on the embedded device was the serialization of the CBOR object, which was solved with modification of the source code after debugging over uART.

4.5.2 Sender

The COSE object is generated in the Sender node. The object follows the proposed structure and for each message a unique initialization vector and a sequence number are generated. An "Encrypt0Message" will be created and each required data field added into their defined header. Different CoAP header protection will be added depending on the transmission request. These values are added as external authentication, also known as additional authenticated data. When the COSE object is finalized it should be encrypted using a pre-shared AES-CCM 128-bit encryption key. With the object encrypted the last step is to serialize the object into a CBOR binary sequence before transmission.

An example implementation for how the COSE implementation in a transmitting sensor may look like can be seen in Appendix A.1.

4.5.3 Receiver

The first step in the receiving node is to verify the integrity of the COSE object. Additional authenticated data will be generated depending on the received header that was encapsulating the CBOR object. The binary sequence is decoded into a COSE object. If the CBOR object has a message tag, when the object will be decoded after that structure. Otherwise, "Encrypt0" will be the default structure. With a COSE object decoded, the generated additional authenticated data will be added to the message. Decryption with a pre-shared AES-CCM 128-bit encryption key will then be performed. If the decryption was successful integrity protection can be ensured, since authentication is provided by encryption in AES-CCM.

An example implementation for how the COSE implementation in a receiving cloud may look like can be seen in Appendix A.2.

5

Evaluation

In this chapter the proposed security architecture is validated for being a reliable and secure end-to-end solution. This was achieved by testing the data flow and is described in the first section. With the implementation validated the proposed solution was analyzed and compared against traditional channel security. This is presented in the last section of the chapter.

5.1 Testing

In this section the implementation is validated by testing the data flow when reading sensor data. The tests were focused in a scenario where one CoAP request result in one CoAP response. The implementation can be seen as a proof-of-concept and thus it was limited to include only GET requests. The reason for testing GET requests only is that this result gives the most complex data flow, where an asynchronous CoAP notification is sent from the gateway after a successfully BLE read is performed. The implementation is validated when an encrypted message sent from the sensor, is successfully decrypted in the cloud and the packet is monitored during transmission. For validation of the COSE support in the sensor these messages were generated in the sensor upon a received read request and wer not hard-coded. The data flow for a CoAP GET request in the system can be seen in figure 5.1.

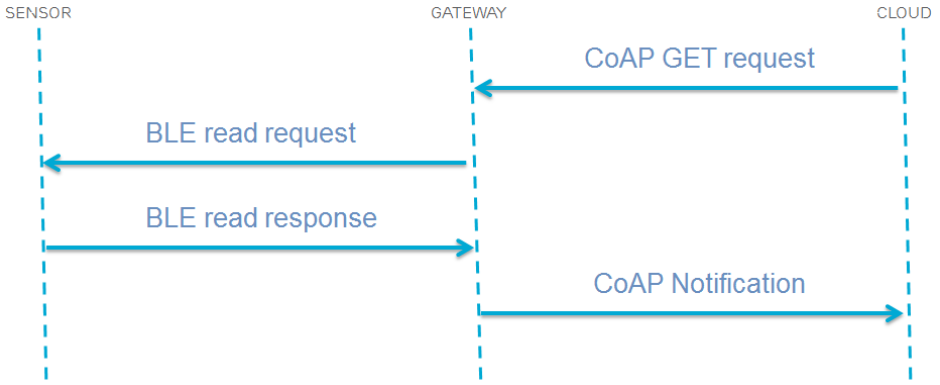

Figure 5.1: *Sequence diagram for a GET request sent from the server.*

When the capillary gateway receives a GET request it initializes a BLE read request to the corresponding GATT characteristic. The sensor then sends a read response with the payload back to the gateway. Due to asynchronous data transmission, a notification is sent up to the server via the observed LWM2M resource, as described in section 4.4.2. This notification is sent as a CoAP Non-Confirmable message.

The data flow was first verified by sending hard-coded CBOR objects as a binary sequence from the sensor up to the cloud. A successful validation of the transmission was made by monitoring the data flow. CBOR decoding in the server was then implemented and by ensuring that decoding of the CBOR object was possible the data flow now had fully support of sending binary formatted CBOR objects. The data flow was monitored over all channels and nodes. Sniffing was performed over the channels for ensuring that the packets had a correct binary format. Debugging tools in the nodes were used for printing and verifying the received data.

With the system and data flow validated the next step was to verify transmission of the proposed COSE object. As mentioned in the methodology chapter the COSE tests were focused in GET requests initialized from the server. During the described data flow the packets are monitored. Validation of the COSE tests was provided when the server successfully decoded and decrypted the received binary sequence. For visualization a demo was produced simulating the data flow and validating the tests. As content a 20-bytes string was sent, which gave a total packet size of 45-bytes due to the overhead. The COSE object was created and encrypted as described in the implementation section, thus ending up with the same structure as in figure 3.2. CBOR serialization was performed in the sensor creating the binary sequence which was printed over UART. The data was then sent over the BLE channel in multiple BLE packets which were monitored using the Texas Dongle and the Texas Packet Sniffer [40].

The packets are visualized in figure 5.2. In the picture it is possible to see the initial read request sent from the gateway at the top. This is followed by multiple read response packets which holds the binary sequence. Data signing is not used due to the mentioned sensor SDK limitation described in section 4.2.3.

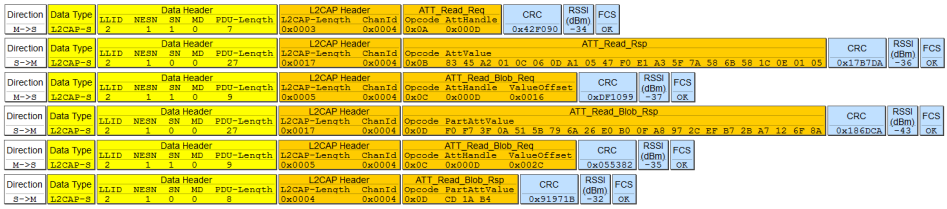

Figure 5.2: *Visualization of the BLE read using Texas Packet Sniffer [40]*

When the packets reach the gateway it is printed with the logger and mapped to the used IPSO object. A CoAP notification is sent to the server and the transmission over the channel can be sniffed using Wireshark or any other packet sniffer supporting IP. The CoAP notification is received by the server and verified by a successful decryption which is printed with the logger. The server console after a successful transmission can be seen in figure 5.3.

```
maj 12, 2017 2:52:05 EM org.eclipse.californium.core.CoapServer start
INFO: Starting server
maj 12, 2017 2:52:05 EM org.eclipse.californium.core.network.CoapEndpoint start
INFO: Starting endpoint at 0.0.0.0/0.0.0.0:5683
maj 12, 2017 2:52:05 EM org.eclipse.californium.core.network.CoapEndpoint start
INFO: Starting endpoint at 0.0.0.0/0.0.0.0:5684
maj 12, 2017 2:52:05 EM org.eclipse.californium.scandium.DTLSConnector start
INFO: DTLS connector listening on [0.0.0.0/0.0.0.0:5684] with MTU [1 280] using (inbound) datagram buffer size [16 474 bytes]
[2017-05-12 14:52:05,737 INFO LeshanServer] LWM2M server started at coap://0.0.0.0/0.0.0.0:5683, coaps://0.0.0.0/0.0.0.0:5684.
[2017-05-12 14:52:05,740 INFO Server] jetty-9.2.11.v20150529
[2017-05-12 14:52:05,794 INFO StandardDescriptorProcessor] NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
[2017-05-12 14:52:05,805 INFO ContextHandler] Started o.e.j.w.WebAppContext@ed7f8b4{/,file:/C:/Users/ehamtja/Desktop/MasterThesis/Implementation/Server/target/classes
[2017-05-12 14:52:05,853 INFO AbstractConnector] Started ServerConnector@524d6d96{HTTP/1.1}{0.0.0.0:8080}
[2017-05-12 14:52:05,854 INFO Server] Started @1224ms
[2017-05-12 14:52:05,854 INFO ServerLwm2m] Web server started at http://150.132.88.177:8080/.
[2017-05-12 14:52:19,205 INFO CoapMessageTracer] CoAP Receive REQUEST.
[2017-05-12 14:52:19,251 INFO EventServlet$1] Client with ID: Gu4uBHW5v8 succesfully registered to server
[2017-05-12 14:52:19,252 INFO EventServlet$1] Trying to subscribe to 15003/0/1
[2017-05-12 14:52:19,483 INFO CoapMessageTracer] CoAP Receive RESPONSE.
[2017-05-12 14:52:19,484 INFO CoapMessageTracer] Notification: true
[2017-05-12 14:52:19,486 INFO CoapMessageTracer] Code: 2.05 Type: ACK
[2017-05-12 14:52:19,508 INFO EventServlet$2] A new observeration has been set with path: /15003/0/1
[2017-05-12 14:52:20,582 INFO CoapMessageTracer] CoAP Receive RESPONSE.
[2017-05-12 14:52:20,583 INFO CoapMessageTracer] Notification: true
[2017-05-12 14:52:20,584 INFO CoapMessageTracer] Code: 2.05 Type: NON
[2017-05-12 14:52:20,598 INFO CoseImpl] Size payload: 45 value: [0@2a5a55e3
[2017-05-12 14:52:20,654 INFO CoseImpl] CBORObject received: [h'A2060D010C', {5: h'F0E1A35F7A586B'}, h'0E0105F0F73F0A515B796A26E0B00FA8972CEFB71DFFD246EB087509']
[2017-05-12 14:52:20,700 INFO Encrypt0Handler] CBORObject decoded!
[2017-05-12 14:52:20,700 INFO CoseImpl] External data: NON1.2
[2017-05-12 14:52:20,714 INFO Encrypt0Handler] Decrypted content: This is some content
[2017-05-12 14:52:20,714 INFO CoseImpl] Authentication successful
```

Figure 5.3: *Server console visualization after a successful transmission.*

5.2 Efficiency analysis

With the implementation validated the proposed solution was analyzed and compared against traditional channel security. In this section an efficiency comparison between the proposed end-to-end security enhancement and current security guidelines is presented. The main focus of the analysis is on packet overhead and since the proposed architecture utilizes the same cryptology solution as the current guidelines, a reliable overhead comparison can be made. A sufficient efficiency analysis would benefit from including an physical comparison regarding workload, CPU time, memory usage and energy consumption etc. In this research no such analysis could be performed, since an implementation based on current channel security guidelines is absent.

5.2.1 Bluetooth Low Energy overhead

The default size for one BLE data packet is 27 bytes. This is the minimum supported size and it can be increased up to 512 bytes, if supported by the hardware [6]. The packet header is included in the used size and therefore the actual payload will be slightly less. In the default case the actual payload is around 22 bytes due to header fields. The default size was used in the validation of the proof-of-concept implementation.

Both security modes in Bluetooth Low Energy applies integrity protection to the packet without affecting the data PDU. They do not increase the number of packet required to be sent, only increases the packet length. As described in the theory section there will be four bytes overhead when using security mode 1 and twelve bytes overhead when using security mode 2, which can be seen in figure 5.4.

Figure 5.4: *Integrity protection overhead for both LE security modes*

When the proposed initial COSE message is used an overhead of 25 bytes is added, thus taking up one default sized BLE data packet. The next COSE messages will have an overhead size of 16 bytes, which results in only one transmitted BLE data packet if the sensor value has a size of 6 bytes or less. For sensor data larger than 6 bytes an extra packet is required. The worst case scenario is that one extra BLE packet may be sent, since the COSE overhead is fixed regardless the number of sent BLE data packets. With the proposed solution it should therefore be more suitable to increase the size of the BLE data packet to the maximum supported, which would allow a much larger payload to be send in one packet.

5.2.2 Constrained Application Protocol overhead

Over CoAP the proposed COSE architecture has slightly less overhead. Equivalent security strength will be achieved even if DTLS has an increased overhead size. The table in 5.1 shows that the proposed COSE structure has less overhead compared to DTLS. This applies for both the initial COSE message structure, declared as COSE 1, and the COSE messages sent after an initialization vector has been exchanged, represented as COSE 2.

Protocol	Overhead
COSE 1	25
COSE 2	16
DTLS	29

Table 5.1: *CoAP overhead size in bytes*

The total packet size for different payloads can be visualized in figure 5.5. The correlation between the payload size and packet size is linear as expected, which can be verified in a recent paper comparing COSE and DTLS overheads [8].

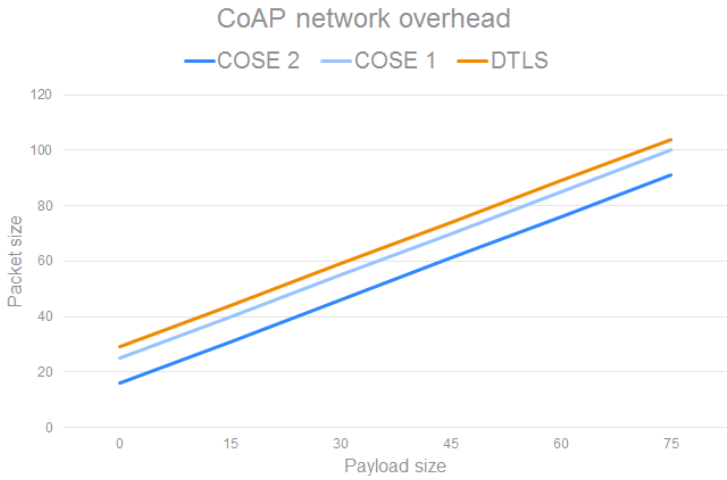

Figure 5.5: *Response packet size for different payloads*

Less overhead will result in less energy consumption, which is desirable for constrained environments. There will be no expensive handshake procedure when using object security, which comes with DTLS for setting up a link layer encryption. There are cases where the DTLS-PSK handshake procedure requires up to 750 bytes being sent between the nodes for setting up the encryption [8].

However, in those cases a Diffie-Hellman key exchange is made which gives each session a unique encryption key. The encryption will therefore ensure better encryption than using COSE, since no session key is used in the proposed solution. A better comparison would be to instead propose a key exchange mechanism for object security, which then could be compared against the DTLS handshake procedure.

5.2.3 Overall efficiency

No physical efficiency analysis was made in this thesis. Previous analyses have shown that the CPU time slightly increases when performing cryptograpic operations for COSE as described in section 2.7. Since the cryptographic support was provided by mbedTLS which is the same library used in this research, similar results should be given for the proposed solution. The cryptographic operations performed in a BLE based sensor should result in the same CPU time as if the sensor used IP. The same cryptographic library is used and in both cases the operations are performed on the application layer, thus different radio channel will not affect the CPU time in that case.

The given results from the studied papers in section 2.7 had an increased RAM and ROM usage due to a large amount of unused source code. In the implementation of the sensor image this was taken into consideration, thus the implemented sensor image, in this thesis, likely has a slightly better efficiency compared to the studied paper. There is for example only support for AES-CCM from the mbedTLS library, thus a small part of the library are actually compiled. Additionally, only the used parts of the COSE-C library is actually flashed to the sensor which decreases the RAM usage. This will however not change the fact that COSE is likely still less efficient than the corresponding link-layer security when performing its cryptographic operations.

6

Discussion

6.1 Research Questions

The research questions have been divided up into three areas. The first question requires an object security solution for the described IoT platform to be proposed. The design is later implemented and tested, which validates the second question. An efficiency comparison focused in overhead is then performed for answering the final question.

6.1.1 Architecture

The proposed architecture was based on the requirement that encryption keys for the COSE object should not be known by the gateway. Authenticated encryption was used which made it impossible to verify the integrity over the gateway without an additional integrity protection. Data signing ensured this protection when reading sensor data over the BLE channel. If the server instead would perform a write operation, then a similar integrity protection as data signing needs to be used over the CoAP channel. The same concept should be used in both directions.

6.1.2 Implementation

The research concluded that Object security and COSE is implementable in practice for a IoT platform based on Bluetooth Low Energy. Some of the biggest implementation challenges were in the sensor. Modifications of the COSE-C library was required for supporting the BLE sensor image. Generation of the COSE object and the BLE GATT profile were also challenging and required a deeper knowledge of the protocols. An overall challenge was the implementation of the data

flow and routing in the IoT platform which enabled support for sending CBOR objects.

The performed tests verified the implementation of both the platform and the COSE packet flow in a good way. Therefore, the second research question is validated and object security is proved to be implementable in practice for the described IoT platform.

6.1.3 Efficiency analysis

The proposed solution ensures the same security strength as traditional channel security. Both LE security mode 1 and DTLS use the same security algorithm, which made it possible to propose an object security solution with equal strength. The overhead comparison over CoAP showed that less overhead is required when using object security compared to DTLS. As a result, this will decrease the energy consumption which is favourable for constrained environments. The main reason for that DTLS has larger overhead is due to the increased sequence number size, and that a new initialization vector is sent for each packet, as described in section 2.3.4. The overhead comparison over BLE concluded that similar energy consumption and access control will be ensured when using LE security mode 2 instead of LE security mode 1. The overhead for the COSE object should not be a problem, since it is possible to increase the BLE data packet size up to 512 bytes which should be sufficient.

The proposed architecture based on object security will however add extra overhead with the use of an extra integrity protection. The COSE object will only ensure security end-to-end, thus the first transmission channel must utilize data signing or similar mechanisms.

It was not possible to compare the run-time efficiency of the proposed architecture against current security architecture due to the abcence of both implementations. Previously studies has shown that application-based security with COSE slightly increases the CPU-time for performing cryptographic operations compared to link-layer based channel security. However, when object security is used the encryption is only performed once and not for each new channel. The energy consumption will therefore in total be less when using object security for ensuring confidentiality over the IoT platform.

6.2 Conclusion

The proof-of-concept implementation validates that object security and COSE can be utilized over a short-range radio network such as Bluetooth Low Energy. End-to-end security for the described IoT platform can be ensured with the proposed architecture and object security. Extra overhead for integrity protection is added due to the requirement that the capillary gateway should not know any keys for the actual COSE object. Overall there will still be less overhead which as a results will decrease the required energy consumption, which is suitable for constrained node networks. Additionally, when using object security the capillary gateway can be lightweight since the data being forwarded is protected at the application layer.

Object security and COSE should be seen as an alternative for ensuring end-to-end security for the Internet of Things.

6.3 Future works

There is room for improvement since the research mainly should be seen as a proof-of-concept validating the usage of object security. It is an interesting concept, but more analysis needs to be performed for validating the usage of COSE.

Instead of using built-in security mechanism such as data signing over BLE, it would be more generic to propose an architecture entirely based on object security. Such a solution could be used over other radio technologies as well.

No key exchange procedure was used and the architecture used pre-shared keys for ensuring security. COSE has support for key-exchange mechanisms but due to the scope of the thesis these were not used. For future work a key exchange analysis could be performed. The OSCoAP specification utilizes key exchange methods which could be studied for this purpose.

Appendix

A

Code examples

A.1 Object Security sender example

The following source code comes from the proof-of-concept implementation for sending a COSE object from the sensor, thus the implementation is based in C:

```
218
219     //Init Encrypt0 COSE object.
220     HCOSE_ENCRYPT hEncObj = COSE_Encrypt_Init(0, CBOR_CONTEXT_PARAM_COMMA NULL);
221     if (hEncObj == NULL) {
222       return -1;
223     }
224
225     //Add COSE Header Algorithm (protected)
226     if(!COSE_Encrypt_map_put_int(hEncObj, COSE_Header_Algorithm,
227       cn_cbor_int_create(COSE_Algorithm_AES_CCM_64_64_128, CBOR_CONTEXT_PARAM_COMMA NULL),
228     COSE_PROTECT_ONLY, NULL)) {
229       return -1;
230     }
231
232     //Add SequenceNumber into Partial IV (unprotected)
233     int sequence_number = 13;
234     if(!COSE_Encrypt_map_put_int(hEncObj, COSE_Header_Partial_IV,
235       cn_cbor_int_create(sequence_number, CBOR_CONTEXT_PARAM_COMMA NULL), COSE_PROTECT_ONLY, NULL)) {
236       return -1;
237     }
238
239     //Set content.
240     char * sz = "This is some content";
241     if(!COSE_Encrypt_SetContent(hEncObj, (byte *) sz, strlen(sz), NULL)) {
242       return -1;
243     }
244
245     //Set external data to be authenticated. (CoAP operation and CoAP version)
246     const byte extdata2[6] = {'N', 'O', 'N', '1', '.', '2'};
247     if(!COSE_Encrypt_SetExternal(hEncObj, extdata2, sizeof(extdata2), NULL)) {
248       return -1;
249     }
250
251     //Encrypt COSE object
252     if(!COSE_Encrypt_encrypt(hEncObj, rgbSecret, cbSecret, NULL)) {
253       printf("\r\n Encryption FAILED: %i", err_callback->err);
254       return -1;
255     }
256
257     //Encode COSE Object into binary sequence format.
258     cb = COSE_Encode((HCOSE)hEncObj, RgbDontUse, 0, sizeof(RgbDontUse));  //Allocate size of the array, cb.
259     if(cb < 1) {
260       printf("\r\n COSE_Encode FAILED.");
261       return -1;
262     }
263
```

A.2 Object Security receiver example

The following source code comes from the proof-of-concept implementation for receiving a COSE object to the cloud, thus the implementation is based in Java:

```
29          //Decode binary sequence into CBOR object
30          CBORObject cborobj = CBORObject.DecodeFromBytes(payload);
31
32          //Generic handler created depending on COSE TAG
33          MessageHandler mMessageHandler = getMessageHandler(cborobj);
34
35          //Decode CBORObject into COSE object given from generic handler
36          mMessageHandler.decode(cborobj);
37
38          //Attach external data to the decoded COSE object (Received CoAP message)
39          mMessageHandler.setExternalData(CoAPMessageOperation, CoAPMessageVersion);
40
41          try {
42              //Decrypt COSE object and get content in bytes
43              byte[] content = mMessageHandler.decrypt();
44
45          } catch (InvalidCipherTextException e) {
46              //Message integrity failure
47              e.printStackTrace();
48          }
```

Bibliography

[1] Ericsson. Ericsson mobility report. page 33, November 2016. Cited on page 1.

[2] Bluetooth SIG. Internet gateways. *Bluetooth White Paper*, February 2016. URL `https://www.bluetooth.com/develop-with-bluetooth/white-papers`. Cited on page 1.

[3] Z. Shelby K. Hartke and C. Bormann. The constrained application protocol. Standards Track RFC 7252, IETF, June 2014. Cited on pages 1, 11, and 13.

[4] J. Schaad. Constrained object signing and encryption (cose). Internet-draft Work in progress, IETF, November 2016. Cited on pages 2, 3, 12, 16, 17, 18, 23, and 26.

[5] Xiaolei Dong Jun Zhou, Zhenfu Cao. Security and privacy for cloud-based iot: Challenges. *IEEE Communications Magazine*, 55(1):26–33, 2017. Cited on page 2.

[6] Bluetooth SIG. Bluetooth core specification version 5. Adopted standard, December 2016. Cited on pages 3, 6, 9, 10, 23, 24, and 36.

[7] E. Rescorla. Datagram transport layer security. Standards Track RFC 4347, IETF, April 2006. Cited on pages 3, 13, and 23.

[8] Joakim Brorsson and Martin Gunnarsson. Compact object security for the internet of things. Master's thesis, Lund University, June 2016. Cited on pages 3, 4, 5, 18, 19, and 37.

[9] S. Kumar P.Moreno-Sanchez F. Vidal-Meca O. Garcia-Morchon, S. L. Keoh and J. H. Ziegeldorf. Securing the ip-based internet of things with hip and dtls. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 119–124. ACM WiSec, April 2013. Cited on page 3.

[10] T. Bray. The javascript object notation (json) data interchange format. Standards Track RFC 7159, IETF, March 2014. Cited on pages 3 and 15.

[11] Ericsson AB SICS Swedish ICT. G. Selander, J.Mattson. Object security of coap (oscoap). Work in progress, IETF, October 2016. Cited on pages 4, 11, 21, 23, 24, 25, and 26.

[12] Malisa Vucinic et al. Oscar: Object security for the internet of things. *Ad Hoc Networks*, 32(3-16), 2015. Cited on pages 4, 21, 23, 25, and 26.

[13] B.Korver E. Rescorla. Guidelines for writing rfc text on security onsiderations. Technical Report RFC 3552, IAB, July 2003. Cited on page 5.

[14] A. Keranen C. Bormann, M. Ersue. Terminology for constrained-node networks. Technical Report RFC 7228 (Informational), IETF, May 2014. Cited on page 6.

[15] J. Postel. User datagram protocol. Technical Report RFC 768, IETF, Aug 1980. Cited on page 6.

[16] Rolf H. Weber. Internet of things - new security and privacy challenges. *Computer Law & Security review*, 26(1):23–30, 2010. Cited on page 6.

[17] Akiba & Robert Davidson Kevin Townsend, Carles Cufi. *Getting started with Bluetooth Low Energy*, volume 1. O'Reilly, May 2014. Cited on pages 6 and 8.

[18] P. Leach et. al. A universally unique identifier (uuid) urn namespace. Technical Report RFC 4122, IETF, July 2005. Cited on page 7.

[19] D.Whiting et al. Counter with cbc-mac(ccm). Technical Report Informational 3610, IETF, September 2003. Cited on pages 8 and 14.

[20] R. T. Fielding. Architectural styles and the design of network-based software architectures. Master's thesis, University of California, 2000. Cited on page 11.

[21] J. Postel. Transmission control protocol. Technical Report RFC 793. Updated by RFCs 1122, 3168,6093, 6528, IETF, Sept 1981. Cited on pages 11 and 13.

[22] Mohit Sethi. Security in smart object networks. Master's thesis, Aalto University, June 2012. Cited on page 11.

[23] T. Dierks. The transport layer security (tls) protocol version .2. Standards Track RFC 5246, IETF, August 2008. Cited on pages 13 and 14.

[24] Babar Shah Abdul Rahman. Security analysis of iot protocols: A focus in coap. In *ICBDSC, 3rd MEC International Conference*. IEEE, March 2016. Cited on page 13.

[25] Daeyoung Kim Jiyong Han, Minkeun Ha. Practical security analysis for the constrained node networks> focusing on the dtls protocol. In *5th International Conference on Internet of Things (IoT)*. IEEE, October 2015. Cited on pages 13 and 14.

[26] T. Fossati Nokia H. Tschofening, ARM. Tls/dtls profiles for the internet of things. Technical Report RFC 7925, IETF, July 2016. Cited on pages 13, 14, and 23.

[27] Open Mobile Alliance, Accessed: 2017-01-31. Cited on page 14.

[28] Christian Bonnet Soumya Kanti Datta. A lightweight framework for efficient m2m device management in one m2m architecture. In *Recent Advances in Internet of Things (RIoT), 2015 International Conference*. IEEE, April 2015. Cited on page 14.

[29] Eclipse Leshan. `https://www.eclipse.org/leshan/`, Accessed: 2017-04-11. Cited on page 14.

[30] Chetan Deshpande Vishwas Lakkundi Suhas Rao, Devaiah Chendanda. Implementing lwm2m in constrained iot devices. In *2015 IEEE Conference on Wireless Sensors*. IEEE, August 2015. Cited on page 14.

[31] Martin E. Hellman Whitfield Diffie. Privacy and authentication: An introduction to cryptography. In *Proceedings of the IEEE, VOL. 67*. IEEE, March 1979. Cited on page 14.

[32] D. McGrew. An interface and algorithms for authenticated encryption. Technical Report RFC 5116, IETF, Jan 2008. Cited on page 14.

[33] F. Palombini. Object security of coap (oscoap). Master's thesis, KTH Royal Institute of Technology, June 2015. Cited on pages 14, 15, and 18.

[34] C. Bormann and P.Hoffman. Concise binary object representation. Standards Track RFC 7049, IETF, June 2014. Cited on page 15.

[35] National Institute of Standards and Technology. Announcing the advanced encryption standard (aes). Technical Report Processing Standards Publication 197, NIST, November 2001. Cited on page 16.

[36] Federal Information Processing Standards. Digital signature standard (dss). Technical Report FIPS PUB 186-4, U.S Department of Commence, July 2013. Cited on page 17.

[37] Phillip Rogaway Mihir Bellare, Joe Killian. The security of the cipher block chaining message authentication code. *Journal of Computer and Sicence Sciences*, 61(3), December 2000. Cited on page 17.

[38] R. Housley D. Whiting and N. Ferguson. Counter with cbc-mac. Technical Report RFC 3610, IETF, Sept 2003. Cited on page 18.

[39] Eclipse Kura. `https://www.eclipse.org/kura/`, Accessed: 2017-04-18. Cited on page 30.

[40] Texas Packet Sniffer. `https://www.ti.com/tool/packet-sniffer`, Accessed: 2017-04-18. Cited on pages 34 and 35.