DEGREE PROJECT IN COMMUNICATION SYSTEMS, SECOND LEVEL
STOCKHOLM, SWEDEN 2014

Lightweight Message Authentication for the Internet of Things

RIKARD HÖGLUND

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Lightweight Message Authentication for the Internet of Things

Rikard Höglund

2014-11-24

Master's Thesis

Examiner and academic adviser
Professor Gerald Q. Maguire Jr.

School of Information and Communication Technology (ICT)
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

During the last decade, the number of devices capable of connecting to the Internet has grown enormously. The Internet of Things describes a scenario where Internet connected devices are ubiquitous and even the smallest device has a connection to the Internet. Many of these devices will be running on constrained platforms with limited power and computing resources. Implementing protocols that are both secure and resource efficient is challenging. Current protocols have generally been designed for mains powered devices; hence, they are not optimized for running on constrained devices. The Constrained Application Protocol (CoAP) is a protocol for network communication specifically designed for constrained devices. This thesis project examines CoAP and presents an extension that adds authentication in a way that is suitable for constrained devices, with respect to minimizing resource use. The proposed solution has been compared and contrasted with other alternatives for authentication, particularly those alternatives used with CoAP. It has also been implemented in code and experimentally evaluated with regards to performance versus vanilla CoAP.

The main goal of this project is to implement a lightweight authentication extension for CoAP to be deployed and evaluated on constrained devices. This extension, called Short Message Authentication ChecK (SMACK), can be used on devices that require a method for secure authentication of messages while using only limited power. The main goal of the extension is to protect against battery exhaustion and denial of sleep attacks. Other benefits are that the extension adds no additional overhead when compared with the packet structure described in the latest CoAP specification. Minimizing overhead is important since some constrained networks may only support low bandwidth communication.

Keywords:

Constrained Application Protocol, CoAP, Internet of Things, message authentication, constrained devices, Contiki, Short Message Authentication Check, SMACK

Sammanfattning

Under det senaste århundradet har antalet enheter som kan ansluta sig till Internet ökat enormt. ”The Internet of Things” beskriver ett scenario där Internet-anslutna enheter är närvarande överallt och även den minsta enhet har en uppkoppling till Internet. Många av dessa enheter kommer att vara begränsade plattformar med restriktioner på både kraft- och beräkningsresurser. Att implementera protokoll som både är säkra och resurseffektiva är en utmaning. Tillgängliga protokoll har i regel varit designade för enheter med anslutning till det fasta kraftnätet; på grund av detta är de inte optimerade för att köras på begränsade plattformar. Constrained Application Protocol (CoAP) är ett protokoll för nätverkskommunikation speciellt framtaget för begränsade plattformar. Denna uppsats undersöker CoAP protokollet och presenterar ett tillägg som erbjuder autentisering på ett sätt som passar begränsade plattformar, med avseende på att minimera resursanvändning. Den föreslagna lösningen har blivit beskriven och jämförd med andra alternativ för autentisering, speciellt de alternativ som används med CoAP. Lösningen har också implementerats i kod och blivit experimentellt utvärderad när det gäller prestanda jämfört med standardversionen av CoAP.

Det huvudsakliga målet för detta projekt är att implementera en lättviktslösning för autentisering till CoAP som ska installeras och utvärderas på begränsade plattformar. Detta tillägg, Short Message Authentication checK (SMACK), kan användas på enheter som behöver en metod för säker autentisering av meddelanden samtidigt som kraftåtgången hålls låg. Huvudmålet för detta tillägg är att skydda mot batteridräneringsattacker och attacker som hindrar en enhet från att gå i viloläge. Andra fördelar är att tillägget inte kräver någon extra dataanvändning jämfört med paketstrukturen som beskrivs i den senaste CoAP-specifikationen. Att minimera overhead i kommunikationsprotokoll är viktigt eftersom vissa begränsade nätverk endast stödjer kommunikation över låg bandbredd.

Nyckelord:

Constrained Application Protocol, CoAP, Internet of Things, meddelandeautentisering, begränsade plattformar, Contiki, Short Message Authentication Check, SMACK

Table of contents

Abstract i
Sammanfattning iii
Table of contents v
List of Figures vii
List of Tables ix
List of acronyms and abbreviations xi
1 Introduction 1
1.1 General introduction to the area 1
1.2 Problem definition 2
1.3 Goals 3
1.4 Research methodology 3
1.5 Delimitations 3
1.6 Structure of the thesis 4
2 Background 5
2.1 Constrained Devices 5
2.2 CoAP 6
2.2.1 REST 8
2.2.2 CoAP Security options 8
2.2.3 CoAP Granular security 9
2.3 Californium 10
2.4 Maven 10
2.5 Contiki 10
2.5.1 Erbium 10
2.5.2 Instant Contiki and Cooja 11
2.6 6LoWPAN 12
2.6.1 Authentication and encryption built into IEEE 802.15.4 as used by 6LoWPAN 12
2.6.2 Exploiting IEEE 802.15.4 encryption and authentication 13
2.7 IPsec 14
2.8 Secure Real-time Protocol 14
2.9 Multimedia Internet KEYing (MIKEY) 15
2.10 DTLS 16
2.11 Lithe: Lightweight Secure CoAP for the Internet of Things 17
2.12 Analysis of Existing Internet Protocols for the Internet of Things 18
3 SMACK 21
3.1 Overview 21
3.2 Keys 22
3.3 Pseudo Random Function 23
3.4 Configuration values 24
3.5 Galois fields 25
3.6 Replay detection 26
3.7 Example scenario 29
4 Method 31
4.1 Hardware 31
4.2 Software environment used for development 32
4.3 SMACK C implementation 32
4.4 Energest 33

5 Analysis 35
5.1 Functional Testing 35
5.2 Comparison of packet overhead 36
5.3 Performance Testing 36
5.4 Testing on other constrained devices 37
5.5 Chapter summary 37
6 Conclusions and Future work 39
6.1 Conclusions 39
6.2 Future work 40
6.3 Required reflections 41
References 43
Appendix A. Detailed results 49

List of Figures

Figure 2-1: Texas Instruments CC2538DK 6
Figure 2-2: CoAP Packet structure [12] 7
Figure 3-1: Keys used by SMACK 22
Figure 3-2: Message sections 26
Figure 3-3: Packet processing flowchart 28
Figure 3-4: Communication steps 30

List of Tables

Table 2-1: Fields of a CoAP packet 7
Table 2-2: Yegin CoAP Security Options fields 9
Table 2-3: Granular security options 9
Table 3-1: Generation of Keys A, B, C 23
Table 3-2: SMACK key values 25
Table 4-1: CC2538 test hardware key values 32
Table 4-2: Energest metrics 33
Table 5-1: SMACK measurements 36
Table 5-2: Vanilla CoAP measurements 37
Table 5-3: Energy statistics 37

Appendix table A-1: SMACK full request 1st transaction (A) 49
Appendix table A-2: SMACK MAC check 1st transaction (B) 50
Appendix table A-3: SMACK full request steady-state (C) 51
Appendix table A-4: SMACK MAC check steady-state (D) 52
Appendix table A-5: Vanilla CoAP full request 1st transaction (E) 53
Appendix table A-6: Vanilla CoAP full request steady-state (F) 54

List of acronyms and abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
AB	Aktiebolag (Joint-stock company)
ACL	Access Control List
AES	Advanced Encryption Standard
AP	Access Point
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CCS	Code Composer Studio
CEO	Corporate Executive Officer
CLI	Command Line Interface
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CTR	Counter
DNS	Domain Name System
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
EU	European Union
GNU	GNU's Not Unix!
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
ID	Identification
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IoT	Internet of Things
IP	Internet Protocol
IPv4	IP version 4
IPv6	IP version 6
JTAG	Joint Test Action Group
JVM	Java Virtual Machine

KDC	Key Distribution Center
KTH	Kungliga Tekniska Högskolan (KTH Royal Institute of Technology)
LPM	Low Power Mode
M2M	Machine to Machine
MAC	Message Authentication Code or Media Access Control depending upon context
MHz	Megahertz
MID	Message ID
MIKEY	Multimedia Internet KEYing
MITM	Man-In-The-Middle
MTU	Maximum Transfer Unit
NSA	National Security Agency
OS	Operating System
PC	Personal Computer
PCAP	Packet CAPture
PKI	Public Key Infrastructure
PRF	Pseudo Random Function
PRNG	Pseudo-Random Number Generator
RAM	Random Access Memory
RC4	Rivest Cipher 4
REST	Representational State Transfer
RFC	Request for Comments
ROM	Read Only Memory
RTC	Real Time Clock
RTP	Real-Time Protocol
RTSP	Real Time Streaming Protocol
SDP	Session Description Protocol
SHA	Secure Hash Algorithm
SICS	SICS (Swedish Institute of Computer Science) Swedish ICT AB
SIP	Session Initiation Protocol
SMACK	Short Message Authentication ChecK
SOAP	Simple Object Access Protocol
SRTP	Secure Real-Time Protocol
S/MIME	Secure/Multipurpose Internet Mail Extensions
TCP	Transmission Control Protocol

TI	Texas Instruments
TKL	ToKen Length
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
VM	Virtual Machine
VoIP	Voice over IP
VPN	Virtual Private Network
WWW	World-Wide Web
XML	eXtensible Markup Language

1 Introduction

This report details the results of a master's thesis project “Lightweight Message Authentication for the Internet of Things” performed during the spring of 2014 at KTH Royal Institute of Technology. The project was conducted in cooperation with SICS Swedish ICT AB [1] in Kista.

1.1 General introduction to the area

During the last decade, the growth in the number of Internet enabled devices has been considerable. At the start of this expansion, people typically only owned a few Internet capable devices, typically in the form of personal computers. Today more and more devices have interfaces that allow Internet connectivity. One of the most significant developments has been in the number of smart phone devices. Currently people frequently own many devices that they use interchangeably for Internet access. Every day additional devices join the global Internet, potentially permitting access to or from them by other Internet enabled devices.

The term Internet of Things (IoT) was coined by Kevin Ashton during 1999 [2], although the concept was discussed in scientific literature prior to this time. This term tries to define a future Internet where the growth in the number of device continues and almost all electronic devices have Internet connectivity. This growth is not limited to user-controlled devices, but also includes machine-to-machine (M2M) communication, such as smart sensor systems. All of these Internet connected devices will have a representation in the Internet either in the form of an IP address or some other identifying information. Setting up such an infrastructure has many benefits, including remote monitoring, convenient control of devices owned by an individual, and increasing numbers of automated systems. Estimates of the number of wireless devices connected to the Internet suggest 30 billion devices by 2020 [3]. Even today, IoT has emerged as an area for research and development.

A "constrained device" is a device that has limited resources in terms of processing capacity, memory, or available power. Constrained devices are often used to implement sensor networks and automated systems that utilize M2M communication. The reason these devices are used is that they are small, inexpensive, and can perform the desired function(s), while consuming very little power. The software running on these devices has to be adapted to this constrained environment and ensures sufficient performance *without* requiring high speed processing, large memory capacity, or using excessive power. Creating small IP stacks and similar software have been necessary steps to realize IoT and to allow constrained devices to communicate efficiently via a network. Making IoT devices accessible through the same protocols used in the global Internet is also important when interconnecting these devices to the existing network infrastructures.

Security is another important aspect of the IoT. If all devices have an IP-address and are accessible via the Internet, then security becomes an even more important issue as the number of potential attackers is greatly increased. Authentication is vital to prevent certain types of attacks against these devices and to confirm the validity of messages. For instance, a device can be inundated with messages in order to exhaust its battery supply, thus authentication has to be performed in an efficient manner and properly take into account the device’s limited power, storage, and computing capabilities. Because of this, the protocols used for authentication have to be adapted for these constrained devices and must meet requirements beyond the conventional requirements for mains powered devices.

Combining the fast growth of IoT devices with limited resources and less mature security options means that these constrained devices can become a prime target for attacks. If these issues are overlooked, then sensitive systems including devices controlling people’s homes or industrial applications are at risk. This is especially true if devices such as smart light bulbs, industrial sensors, radiators, and other such applications continue to gain in popularity. It is important that security is built into the IoT as early as possible, as retrofitting security solutions is more a difficult challenge.

1.2 Problem definition

Denial of service (DoS) attacks are malicious actions that attempt to deny access to or shut down some device [4]. For instance, such an attack could attempt to overwhelm a target with traffic or send malformed packets that the device does not know how to handle. For constrained devices the term "denial of sleep" is often used since these devices frequently rely on going to sleep when data is not being actively processed, sent, or received. It is crucial for the radio circuitry to be in sleep mode as long as possible in order to minimize power consumption. Simply receiving, parsing, and processing data sent to a constrained device can be costly since the radio needs to be on in order to receive the data, which drains a lot of power. In addition, power is need for processing the packet(s) that have been received. For these reasons, constrained devices are especially vulnerable to DoS attacks. In addition, they are rarely protected with intrusion detection and prevention systems that are common for servers on the Internet. However, firewalls or other types of gateways can be used to protect these devices from traffic coming from outside the local network.

The problem addressed in this thesis project deals with how to design, implement, and evaluate a message authentication extension to CoAP for constrained devices. This solution must be both secure and economical with the resources of the system. Several important aspects have to be taken into consideration in order to ensure that the proposed extension is secure. First, it should be based on well tested security concepts, i.e., concepts that have been proven over time. Furthermore, authentication must be provided in an efficient manner. When it comes to resource usage, the extension should use code that has been optimized to reduce the amount of memory necessary to ensure that the resulting code will fit into the available memory of the device in question. In addition, the code should be adapted to minimize the required processing power. This can be accomplished by careful selection of algorithms and by reducing the size of fixed arrays and other memory structures that is used. Note that the extension may trade FLASH storage for processing, as generally, microcontrollers have increasing amounts of flash memory – but want to avoid either increasing their processor clock rates or requiring a much larger random access memory (RAM) – as both take additional dynamic power. The amount of RAM available is typical less than the FLASH storage and this limits what session information and other dynamic data that can be allocated and stored in RAM. However, some data has to be kept for each connected device in order to keep track of the session state and which keys are being used.

Short Message Authentication CheckK (SMACK) is an extension of the Constrained Application Protocol (CoAP) [5] specialized for providing message authentication in a resource efficient manner. The methods used to provide this are general and can also be adapted to other communications protocols. Currently a prototype implementation of the SMACK authentication extension to CoAP exists (written in Java). It has been developed internally at SICS as a proof of concept (see beginning of Chapter 3). However, this implementation cannot be tested on most constrained devices since they have insufficient resources to run the required Java virtual machine, hence for such a constrained device there is a need for an optimized implementation in C. This means that a version of this extension written in C needs to be created, tested, and evaluated on constrained devices. This version should be deployed on actual constrained devices for performance testing.

SMACK needs to be discussed in the context of other authentication protocols. An energy efficient authentication protocol should lessen the severity of any DoS, such as a denial of sleep attack, since data that is **not** authenticated could be ignored by the device. Ignoring such data allows devices to reduce unnecessary message processing and only reply to legitimate requests. However, the resources used for authentication can be part of a DoS attack, hence we need to minimize the energy consumed by this authentication process. As there are a number of alternative protocols for authentication and confidentiality that could be used with CoAP, some of these protocols will be described and contrasted to the solution presented in this thesis.

1.3 Goals

The following goals are the main objectives of this project:

1. Implement the SMACK extension to the CoAP protocol in C,
2. Optimize and adapt the implementation to run well on constrained devices,
3. Test and evaluate the performance of the extension on actual constrained devices,
4. Describe other existing alternatives for authentication and how they differ from SMACK,
5. Compare SMACK to vanilla CoAP with practical experiments on constrained devices,
6. Test the system to see that it ensures proper authentication, and
7. Take into account the above test results to improve the extension wherever possible.

1.4 Research methodology

This project has selected a quantitative research methodology because the nature of the topic is suitable for statistical analysis. For instance, the round trip time for a CoAP request with the SMACK extension enabled versus a vanilla CoAP request can be compared. Additionally, we can compare and evaluate the performance of the SMACK extension by collecting data about the time spent performing specific calculations. As the goal of this thesis is to evaluate the SMACK extension to CoAP a quantitative research method is most suitable. The project also considered a qualitative methodology; however, it was rejected as quantitative data and statistical analysis of performance are important. The possibility to automate testing and analyze the results of this testing in a consistent manner favors a quantitative approach.

This project uses a deductive approach to investigate the hypothesis that the SMACK extension to CoAP is a viable option for lightweight message authentication on constrained devices. The basic functionality of the SMACK extension will be tested and verified to work. In addition, the question of to what extent SMACK and in which areas SMACK is superior or inferior to existing systems will be discussed. Existing systems will be evaluated based upon external resources, while SMACK will be analyzed using experimental data.

Empirical research will be performed as a part of this project, thus generating experimental results and data. The project will also utilize secondary sources to acquire information and allow analysis of systems that are not available for direct experiments. Comparison and evaluation of SMACK versus vanilla CoAP will be done using new data collected explicitly for this purpose. Results from these tests will be presented using statistical methods to highlight the performance of the different solutions that have been tested. The hardware used for the tests will be identical and care will be taken to measure equivalent processes relevant to each solution in order to ensure a fair comparison.

1.5 Delimitations

This thesis focuses on issues related to CoAP and authentication protocols implemented at the application, transport, or network layers. There are also methods for authentication and encryption in the lower layers of the protocol stack, such as those built into IEEE 802.15.4 [6] - used by 6LoWPAN. Further details of 6LoWPAN are given in Section 2.6 and relevant details of IEEE 802.15.4 are given in Section 2.6.1. However, these solutions rely on the fact that the underlying network utilizes a specific technology. Most devices connected to the Internet utilize versions of Ethernet that do **not** support similar functionality. In order to make the proposed solution as general as possible, authentication should be implemented at the network or higher layer. Protection at higher levels also provides end-to-end security, which is important for many applications. Additionally, because the IPv4 and IPv6 protocols are so ubiquitous the solution should be compatible with these network layer protocols, thus this compatibility is a minimum requirement. Therefore the proposed solution does not rely on any specific lower layer technologies. However, we can of course learn from the mechanisms that have been applied at these lower layers (see for example Section 2.6.2).

The Contiki operating system (OS) will be used as a basis for the implementation of the authentication extension to CoAP. Section 2.5 gives relevant details of Contiki. Contiki currently has a fully functional CoAP implementation named Erbium (see Section 2.5.1) that is used as a basis for the proposed SMACK implementation. Furthermore, Contiki supports a large number of hardware platforms [7] and is a widely used OS for constrained devices. Contiki is open source software; hence all of the relevant source code is freely accessible. In addition, Contiki has a development environment that includes the Cooja network simulator (see Section 2.5.2). This simulator facilitates testing of applications. The evaluation considers the *feasibility* of the SMACK extension for authentication *independent* of the underlying hardware platform. However, benchmarks are used to understand how this authentication protocol performs in comparison with other potential alternatives when actually running on constrained devices.

1.6 Structure of the thesis

The thesis started with a chapter setting out the problem and goals to be addressed in this thesis project. Chapter 2 provides the background knowledge required to understand the topics discussed in the rest of this thesis. This includes technical background concerning existing protocols for authentication that could be used together with CoAP. Chapter 3 describes the design of SMACK and the details of how performing authenticating with it works. After this, the development environment used and information about how to develop for Contiki are given in Chapter 4. This same chapter describes the methodology and design methods used to create a C implementation of the SMACK protocol. The focus of this chapter is on several of the challenges encountered during the development process. Chapter 5 analyzes the results of the implementation described in the previous chapter and compares the proposed solution with alternatives. The thesis concludes in Chapter 6 with a summary of conclusions, suggestions for future work, and a description of some of the ethical, social, economic, sustainability, and other aspects of this thesis project.

2 Background

This chapter contains background information concerning several concepts and tools used this project. Some of these concepts and tools are described in further detail in later chapters. A key aspect underlying this thesis project is the hardware of the constrained devices on which the authentication software is deployed. Another important topic is the software employed during the development and coding phases of this project. Several alternatives for implementing security on constrained devices exist. Some of these are the same protocols used on the Internet and some are protocols that have been adapted or even custom made to function better on constrained devices. The chapter concludes with a description of a number of different communications protocols and standards, as much of the thesis deals with these protocols. Details of these protocols are necessary in order to consider the solutions that have been chosen.

2.1 Constrained Devices

For the purpose of this thesis project, a constrained device is defined as a device that has limited resources in the form of hardware, such as memory, processing, and power. The available power is frequently limited capacity batteries. Furthermore, constrained devices often utilize networks with limited available bandwidth. Combining these factors means that memory usage, algorithm efficiency, and low bandwidth communications are important issues. As power consumption is reduced in sleep mode, these devices typically rely on rapidly transitioning to different levels of sleep when possible, thereby minimizing their battery power usage. Because some components draw more power than others do, it is especially important to optimize the power management of these components. The radio is normally the component that uses the most power and thus keeping it in sleep mode as much of the time as possible is quite important [8].

Sending data is more costly than doing calculations locally. Measurements by Madden et al. have shown that on some systems transmitting one bit is the equivalent of executing 800 instructions [9]. Because of this, the radio should be in sleep mode as much as possible and communications should be kept to a minimum. Often constrained devices are used as sensors or actuators, they generally have limited to no user interaction. This means that they are to a large extent autonomous, sending and receiving data only as necessary. Such M2M communication is different in character and pattern from user-induced communication. If some part of the system malfunctions or an attacker starts sending data to a node, then the node can be tricked into accepting incorrect data readings and the battery could be rapidly drained by keeping the radio constantly listening. In automated systems this might not be noticed until the battery is exhausted, unless the system is designed to inform a management systems of such an apparent attack.

Figure 2-1 shows a typical example of a constrained device in the form of the Texas Instruments' (TI) CC2538 board attached to a SmartRF board. This combination is frequently used for development purposes. This particular board has a maximum clock speed of 32 MHz [10]. There are constrained devices that are even more limited with regard to resources. For instance the Tmote Sky only has a 8 MHz processor and 10 kB RAM [11], whereas the CC2538 has maximum of 32 kB of RAM (depending on the specific model of this chip). Typically, constrained devices clock the processor at a lower clock rate to reduce power consumption and because they have less demanding computational requirements, thus draining the battery at a slower rate than when clocking the processor at a high rate.

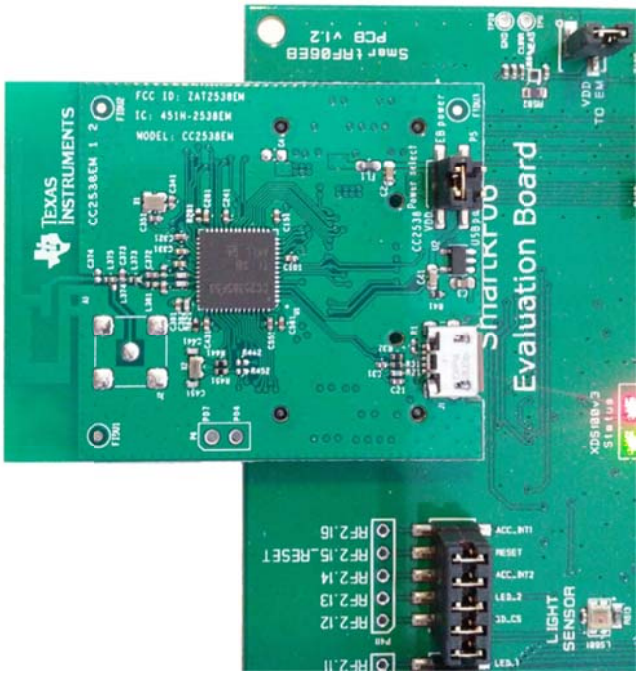

Figure 2-1: Texas Instruments CC2538DK

2.2 CoAP

The Constrained Application Protocol (CoAP) [12] is a communications protocol operating at the application level of the Transport Control Protocol (TCP)/Internet protocol (IP) stack. It is specialized for use with constrained devices that suffer from limited resources, specifically limited network bandwidth or processing power. Key features include low overhead and complexity, unicast and multicast support, optimizations for use on constrained devices, and asynchronous message exchanges [12]. An important use case for CoAP is M2M communication where devices autonomously communicate with each other – normally without human interaction apart from maintenance of the devices. This kind of traffic has different characteristics than communications initiated by a person. For instance, the traffic pattern can differ since humans tend to request resources in a more arbitrary pattern whereas machine communication can be extremely regular. This regularity can also help since it is easier to predict when to expect the next transmission and thus power on/off the radio so as to only operate when communication is expected.

CoAP is quite similar in structure to the Hypertext Transfer Protocol (HTTP), but is less complex. CoAP follows a simple request/response model that allows one device to act as a client, requesting resources, and the other device as a server providing some resources. As with HTTP, Uniform Resource Locators (URL) are used to identify resources. For instance, *coap://example.com/temp* could provide data from a temperature sensor on a device at the host named *example.com*. CoAP also uses a subset of the same methods (in the form of GET, PUT, POST, and DELETE) that HTTP employs. The response codes follow the style of HTTP with a reply containing a resource using the code "2.05 Content" and a reply when no corresponding resource was found uses "4.04 Not Found". As is the case with HTTP, CoAP has many other response codes for various scenarios.

One key difference between HTTP and CoAP is that the transport protocol used is User Datagram Protocol (UDP); in contrast, to HTTP which uses TCP. This choice of transport protocol has some drawbacks, but also provides some benefits. One drawback is that the reliable transmission that TCP guarantees has to be implemented by the application on top of UDP. CoAP uses "confirmable messages" to specify that a message must receive an acknowledgement to confirm that the message arrived correctly. There are also non-confirmable messages that do not require such a confirmation. CoAP can utilize multicast traffic since it marks such messages as non-confirmable and thereby avoids an implosion of acknowledgements. This means that CoAP refrains from creating a communication session with receivers, as is done when using TCP, but rather it builds upon the sessionless functionality UDP provides.

The properties of CoAP are well adapted to constrained devices in that it is a relatively simple protocol with a simple packet structure (see Figure 2-2). Using UDP reduces per packet overhead and by not utilizing the acknowledgements sent by TCP the protocol reduces communication overhead. The ability to specify whether messages require confirmation means that devices can do common actions such as checking a sensor value frequently *without* inducing too much network traffic and can reduce the need to switch the radio on (thus helping to minimize power consumption). Typically, an isolated lost message is *not* crucial since a missed reading will simply mean that the system will use the latest value *until* the next scheduled sensor reading.

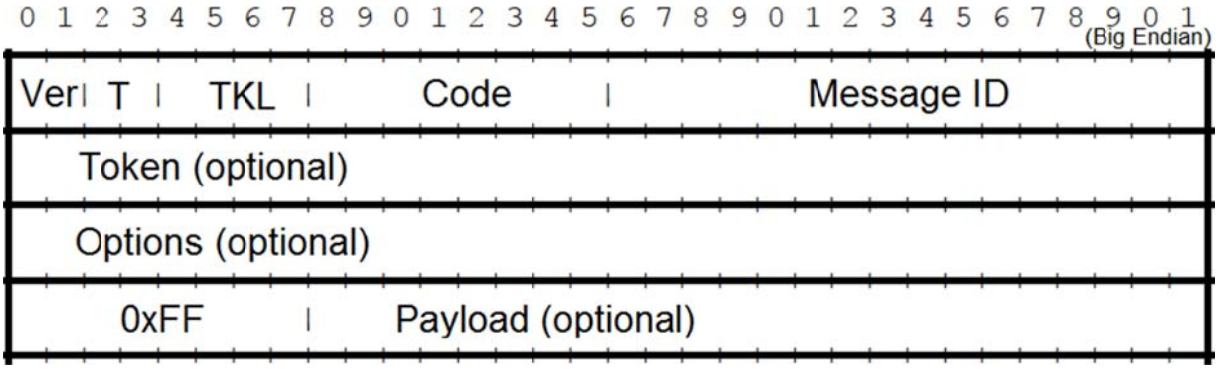

Figure 2-2: CoAP Packet structure [12]

Table 2-1 summarizes the different fields of a typical CoAP packet. This packet structure is simple and has few mandatory fields. The fact that many of the fields are optional makes CoAP a flexible protocol. CoAP allows an application to send small non-confirmable messages with minimum overhead, but an application can also send more complex packets with options (including data) and may require confirmation from the receiver. The size of the mandatory header of a CoAP packet is only 4 bytes. The maximum recommended payload size is 1024 bytes to ensure that the CoAP packet can be sent unfragmented in an IP packet. This assumes an IP Maximum Transfer Unit (MTU) of 1280 bytes. The standard also specifies that assuming lower values may be beneficial, especially on IPv4 networks, as the actual path MTU is difficult to determine with certainty. In addition, 6LoWPAN networks limit link layer frames to only 127 bytes [12].

Table 2-1: Fields of a CoAP packet

Field	Size in bits	Description
Ver	2	Version of CoAP used
T	2	Type of message: confirmable, non-confirmable, acknowledgment, or reset
TKL	4	Length of token field in bytes (0-8 bytes)
Code	8	Message code (such as 4.04)
Message ID	16	Matches request/replies
Token	0-64	Correlate request/response
Options		Similar to options of HTTP Message type, proxy options, cache max age, …
Payload Marker (0xFF)	8	Marks start of payload
Payload		Data

Matthias Kovatsch has contributed a lot to CoAP implementations, work on constrained devices, and specifically Contiki. Both the Erbium CoAP implementation for Contiki and the standalone Californium CoAP Java library were written mainly by him [13]. He is the owner of the Californium GitHub code repository and a Contiki fork featuring Erbium [14]. By design, the CoAP protocol lacks any method for authentication. Instead, the DTLS protocol (described in Section 2.10) is *expected* to be used as the main method for achieving secure communication, including authentication, on top of CoAP. At the time of the start of this thesis project, the best way to implement authentication for CoAP was an open research question. A potential solution utilizing new CoAP options for security is described in Sections 2.2.2 and 2.2.3.

2.2.1 REST

Representational state transfer (REST) is a system architecture used by many Internet protocols, a prominent example being HTTP and the World Wide Web in general. REST was first presented by Roy Fielding in his doctoral dissertation in 2000 [15]. In essence, REST tries to define and describe the different components of systems, including how they can best interact. The world-wide web (WWW) be can be said to be a system based upon URLs that allow objects stored at websites to be accessed using the HTTP protocol with a limited number of operations (GET, PUT, POST, DELETE) to access web content according to fixed rules. The idea behind REST heavily influenced the design of HTTP/1.1 as can be seen in chapter 6 of Fielding's dissertation [15].

One of the central ideas of REST is to rely on specific keywords or operations to perform actions on a resource, typically an URL. One of the main alternatives to REST is Simple Object Access Protocol (SOAP) that uses arbitrary keywords or function names for executing operations. In SOAP, functions are defined using XML and then executed against data [16]. One benefit of the REST architecture is that it is lightweight and less complex because of its restricted set of predefined simple operations. As CoAP is similar to HTTP, CoAP also utilizes the REST paradigm for accessing and publishing resources.

2.2.2 CoAP Security options

In an expired Internet-Draft A. Yegin and Z. Shelby proposed "a set of Constrained Application Protocol (CoAP) options that are used for providing data origin authentication, integrity and replay protection, and encryption for the CoAP messages." [17] They defined a new method for securing CoAP other than the two security protocols (IPsec and DTLS) mentioned in the CoAP specification. According to their draft, different cryptographic algorithms should be supported and it should be possible to negotiate between the sender and receiver which algorithms will be used. The new security functionality is implemented by defining new options for the CoAP header. These new headers are used by the sender to request different security levels and to communicate data that will subsequently be used for the encryption or authentication.

Their draft only mentioned the Counter with CBC-MAC (CCM) mode [18] of the Advanced Encryption Standard (AES) encryption function. In essence CCM combines the Counter (CTR) and Cipher Block Chaining-Message Authentication Code (CBC-MAC) algorithms to provide authenticity and confidentiality to data [19]. Positive aspects of this approach are strong security using the AES cipher, authentication of data, and full encryption of the data. Disadvantages include 30 bytes of additional overhead - since each encrypted/authenticated packet must include the values shown in Table 2-2 (in addition to the normal CoAP headers and payload).

In contrast, the SMACK extension does not require any additional overhead when compared to the CoAP specification that allows for a token field of 0-8 bytes of which SMACK by default only utilizes 4 bytes. As the network links used by the constrained device are often constrained from a bandwidth point of view, it is important to minimize the overhead low, thus minimizing the processing power utilized by the nodes. If the payload of the transmitted messages is small, cases can occur where the overhead is a large majority of the data contained in each sent packet. Sending many small messages, as opposed to a few larger ones, is particularly sensitive to big overhead since the overhead is added for each message.

Table 2-2: Yegin CoAP Security Options fields

Name	Size (bytes)
Context ID	1
Nonce	12
MAC	16
OptionCount	1
Total	30

2.2.3 CoAP Granular security

Another similar approach is proposed in a paper by Granja, Monteiro, and Silva [20]. Their paper identifies some key problems with the existing CoAP-DTLS security solutions. First, the security DTLS provides is applied to all packets in a session and there is no provision for granularity or specifying exactly which packets should be protected. There is also no way to specify different levels of protection or to use different algorithms for certain packets. Finally, the end-to-end transport layer security DTLS provides can interfere with gateways and proxies used with CoAP. Therefore, they propose an application-layer security solution that is more flexible with regard to security options and allows for intermediaries, such as gateways.

The encryption algorithm used to provide security is by default AES-CCM. This algorithm is based on the well-known AES algorithm and uses the CCM mode of operation that provides both authentication and confidentiality. In fact, constrained devices often have built in hardware support for AES and AES is frequently used to protect traffic at the link layer. Many of the existing security proposals for 6LoWPAN use this algorithm (including the security options mentioned in Section 2.2.2).

This approach relies on defining new header options for the CoAP protocol to add security. A SecurityOn option is added to each protected packet. This option specifies the details of the protection to be applied and includes other features, such as timestamps that help to protect against replay attacks. A SecurityToken option is used to provide identity data, including username/password, public keys, certificates, or Kerberos tickets. Another field named SecurityEncap contains authenticity and/or encryption parameters including a nonce value, Message Authentication Code (MAC), and any encrypted data. All of these options are on a per-packet basis. The estimated overhead induced by the different options is shown in Table 2-3.

Table 2-3: Granular security options

Name	Size (bytes)
SecurityOn	30 bytes (assuming 20 byte URI)
SecurityToken	Minimum 1 byte + variable identity data
SecurityEncap	Nonce (12 byte) + optional MAC (8 byte) + variable encrypted data

As can be seen in the table, the amount of overhead depends on the security option(s) selected. The calculations by Granja, Monteiro, and Silva state that overhead can be between 11-55% of an 88 byte payload (the maximum size to avoid 6LoWPAN fragmentation). The proposed solution in their paper is faster when taking advantage of the granular control and protecting only some of the packets. However, if all the payloads are encrypted and signed, then the maximum message rate per second is slower for a given amount of bandwidth than CoAP with DTLS. Their measurements show the energy usage of each solution and notes that their proposed solution is *only* superior when selectively applied to packets (specifically only request/replies) but is slower than CoAP with DTLS if all of the packets are signed and encrypted.

2.3 Californium

Californium is a CoAP implementation written in Java. Matthias Kovatsch, one of the main authors of Erbium (see Section 2.5.1), is the main author of Californium [21]. Californium currently supports up to CoAP draft version 11 and has been tested for compatibility at various so called plugtests that grade the compatibility with the CoAP specification [22]. This implementation follows a typical object oriented architecture with the logic split into several parts and employs abstraction and modularity to a great extent [23]. This implementation also attempts to be backwards compatible with older versions of the CoAP draft, since the protocol is still changing rapidly via frequently released drafts. Although the new drafts normally try to avoid major changes to the protocol's functionality.

The resulting Java library provides a baseline implementation that can be used for further development. One drawback is that the language chosen for this implementation was Java, as this is unsuitable for constrained devices because of the high requirements the Java Virtual Machine (JVM) has. In contrast, for development on a PC or another device that supports Java this implementation provides a highly functional library that can be used as a base for a CoAP application or for extensions. Another drawback is the lack of any security solution, this drawback is mentioned as future work in the report on Californium [23]. By default, messages are sent in clear text and do *not* use any form of authentication. Other systems have to be used to provide security to the transmitted CoAP messages; however later versions of Californium started to implement DTLS support and allow accessing such resources using the "coaps://" designation.

2.4 Maven

Maven is a tool for simplifying the building of Java based applications. In some ways it is similar to the GNU "make" command for compiling software [24]. Maven is based on XML configuration files that define the build process and any dependencies [25]. Californium is distributed in a form that is easily compiled and developed using the Maven utility. This makes it easy for a developer to make changes and quickly recompile the project. This tool is helpful since Californium is quite large and has many interdependent classes as the design goal is to create a modular and layered design solution [23]. Maven also executes a number of built in tests when compiling software, this means that it is easier to find not only obvious errors in the code but also to find subtle problems that cause the test conditions to fail.

2.5 Contiki

Contiki is an operating system specifically created for use on constrained devices. It was initially written at SICS mainly by Adam Dunkels. He is currently CEO of Thingsquare, a company focusing on the Internet of Things. Their business idea is to provide interconnectivity of things, mostly focusing on connecting devices to smartphones in a convenient fashion. Contiki is a very resource conservative operating system and can function on devices with as little as 10 kB of RAM and 30kB Read Only Memory (ROM). Contiki is also known for having a small implementation of the IPv6 stack named uIPv6 that was jointly developed with Cisco and Atmel [26].

Contiki is open source and features other innovative technologies, such as protothreads (a memory efficient way to implement rudimentary threading). Basically protothreads are stackless threads which mix an event driven and linear model of execution [27]. Benefits of these technologies include support for threading whilst maintaining a low memory footprint and also in many cases reduced complexity compared to conventional fully multi-threading systems. Other functionality supported by Contiki is low power radio networking and power profiling [28]. Low power requirements and solid support for sensor networks are additional advantages.

2.5.1 Erbium

There is an implementation of CoAP included with the Contiki source code. This implementation is named Erbium and it supplies a fully functioning version of the CoAP protocol specification written in C. It is written by Matthias Kovatsch, the same author who created the Californium library. Erbium has been tested to ensure that it follows the latest (at the time - 2012) CoAP draft. This implementation takes into account the need to reduce power usage and is presented as a "low power CoAP for

Contiki" [29]. This implementation takes full advantage of other optimizations that exist in Contiki, such as the ContikiMAC mechanism that ensures the radio is switched off for as much time as possible - while still retaining the ability to communicate efficiently. It also uses protothreads in order to improve message handling, support callback functions, and asynchronous reception of packets. The full CoAP implementation for Contiki is around 2600 lines of C code.

Erbium employs the built in REST engine of Contiki allowing developers to easily implement protocols following the REST communication architecture, such as HTTP and CoAP. This reduces Erbium's memory footprint since code reuse is high. Erbium combines an efficient CoAP implementation with the Contiki radio optimizations resulting in a system that can communicate efficiently while having a small memory footprint and offers energy savings of up to 26 times [29] compared to a naive implementation that does not take advantage of the benefits conferred by ContikiMAC.

2.5.2 Instant Contiki and Cooja

A development image file called "Instant Contiki" is provided to facilitate developing software for Contiki. This image file can be loaded into virtualization software, such as VMware [30]. Loading this image immediately provides a developer with all the tools needed to start developing and testing software for Contiki. This image file includes the full Contiki source code and a preconfigured development environment. Instant Contiki is the recommended system for developers to use when working on Contiki development and new applications. The image itself is based on a standard version of the Ubuntu Linux distribution.

Included in the image is a network simulator for simulating a network of nodes running Contiki whilst communicating with each other. The simulator, named Cooja, allows developers to easily test and develop software for the Contiki platform. It supports extracting statistics from nodes and monitoring the network traffic to analyze what data is being sent over the network. Starting Cooja dynamically compiles the entire Contiki source including any changes or additions made. It will then start running the specified applications on the simulated nodes. Cooja can simulate various networks with different applications running on the nodes, for instance a server application on one node and a client on another or more advanced topologies.

By loading a client process on one node and the server on another node interaction between these nodes can easily be simulated and monitored. One of the main benefits of using Cooja is that it simplifyies analysis of the behavior of the executing program(s). For instance, nodes can be monitored as if they were real physical devices. Various ports and input/output data can be monitored from Cooja and there is even a simple Command Line Interface (CLI) available for each node. A major benefit is directly collecting the text output from each node in the simulation. This helps with debugging and ensuring that the program behaves correctly.

Furthermore, Cooja supports sniffing of network traffic generated by the simulation and dumping the captured traffic to external Packet CAPture (PCAP) files that can later be input to programs such as Wireshark. This is useful because this project, and most applications of Contiki, rely on network traffic and communication with other nodes. In this way, the network traffic generated by CoAP nodes running under Contiki can easily be captured and the relevant fields checked to see that they both meet the CoAP requirements and later that they correctly utilize the new extensions SMACK provides. Wireshark has a very mature packet-parsing engine that shows the names and decoded values of all the fields in a packet and Wireshark supports parsing of CoAP messages.

The alternative of analyzing the software on actual hardware is more cumbersome, especially as capturing network traffic can be a problem when the boards are communicating directly with each other via radio. The process of transferring a program to the boards for testing also takes time and slows down repeated cycles of editing and testing that can be necessary when developing software. This is particularly true when the development method relies on making incremental changes and

testing whether the functionality is still correct. Extracting the output from boards frequently requires connecting to them via Universal Serial Bus (USB) and retrieving their output*.

Several configuration files and programming examples are included in Instant Contiki and these can easily be loaded into Cooja. Cooja can simulate both simple client/server topologies and more complicated sensor networks with more than 10 nodes with all of these nodes communicating concurrently. The simulation can be on a high abstract level, i.e., concerned with the application layer. Alternatively, Cooja has the ability to simulate lower levels - since each node has a physical position and the transmission properties of the radio medium can be specified [30]. For this thesis project, the properties of the radio medium are not relevant, as CoAP operates on higher levels of the network stack, thus the simulator is primarily used to evaluate code at the application layer. For this reason interference and signal loss will not be parameters in the simulation.

2.6 6LoWPAN

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) is an adaption of the IPv6 protocol to facilitate operation in low power wireless networks [31]. It relies on the IEEE 802.15.4 standard that specifies the operation on the physical and media access layers [6]. One of the key functions of 6LoWPAN is compression of the IP headers and headers from other protocols (such as UDP) [32][31][30][31][30]. The purpose of this compression is to reduce the bandwidth usage over low power networks that are constrained in terms of bandwidth and time available for transmissions. This compression technique works by splitting communications into separate "contexts" that share common knowledge of IPv6 addresses and other metadata. For instance, the context 0x01 (defined in hexadecimal) could be a replacement and common representation for a specific IPv6 address or even a combination of source and destination IPv6 addresses. In this way only 1 byte will be used for the sender and receiver addresses instead of 32 bytes that would otherwise be necessary when not using header compression [32]. Devices can set up such a context and then rely on the abbreviated, compressed addresses for communication. There are specific adaptions of this compression technique optimized to function well with DTLS [33].

2.6.1 Authentication and encryption built into IEEE 802.15.4 as used by 6LoWPAN

The IEEE 802.15.4 standard provides support for encryption of the transmitted frames. This security enables authentication and encryption at the MAC layer [6]. It also includes support for Access Control Lists (ACL) that can filter which devices are allowed to communicate in the network. It is up to an application to request and set the required security parameters in the MAC layer before transmission. An application can signal to the MAC layer that the next packet to be transmitted needs integrity protection or full confidentiality (via encryption) of the packet's contents. After this, the MAC layer provides this functionality as a service.

An issue when using ACLs is that when one node acts as a gateway for the network, by relaying traffic from outside, there is no way for the internal devices to know from where the traffic originates. When using ACLs the nodes have to either accept all the traffic from the gateway or reject all the traffic from it. When traffic arrives from different computers connected to the Internet the nodes beyond the gateway cannot differentiate between these sources nor can they ascertain if the received packets are authenticated. In contrast, a security solution working on higher layers can provide this differentiation of sources and authentication since a higher layer solution is unaffected by the choice of link layer technology. Higher layer addressing information is also not lost between devices as is the case with the MAC address.

* As constrained devices have limited amounts of RAM it may be difficult to collect large log files on these hardware platforms.

A security analysis presented by Sastry & Wagner [34], found some security problems with how IEEE 802.15.4 implements security. They even state that some of the options available *reduce* the overall security instead of increasing it. Some of the potential security issues listed in their paper are:

- Using the same key for multiple ACL entries,
- Losing ACL state at power failures,
- Poor practical support for group keying,
- Shared keying destroys replay protection,
- Modes that employ encryption but not authentication, and
- Insufficient integrity protection.

One downside of the authentication support built into IEEE 802.15.4 is that it is at the link layer. This means the protection is only for a specific link. For this reason, security associations and keys have to be set up for all devices in the network if end-to-end security is to be guaranteed. Shelby and Bormann argue in their book "The Wireless Embedded Internet" that even with strong link-layer security the data remains vulnerable in certain situations. This is true when data leaves a link, traverses a link with reduced security, or when data is forwarded at the network layer [35]. This is noteworthy because many networks do not use IEEE 802.15.4 all the way from the sender to the receiver. The drawback is that data will be unprotected at some point or that an additional security system will be needed on top of what IEEE 802.15.4 provides. As a result they recommend IPsec as a good layer 3 security solution for LoWPAN networks to complement or replace IEEE 802.15.4.

An example of potential security risks is the case where a node is compromised, as in this case traffic going to the node will be authenticated and traffic coming from it will be authenticated. The problem is what happens to the traffic at the node itself. The problem stems from the lack of end-to-end security, as traffic may be modified at the node but will still appear to other nodes in the network to be coming from an authenticated device. Furthermore, IEEE 802.15.4 supports group keys for authentication in a larger group, thus if these keys are compromised, perhaps by a physical attack on a specific node, then the keys for the entire group will be exposed to the attacker.

Nonetheless, there are advantages to implementing security at layer 2. One is that the entire frame contents following the IEEE 802.15.4 header are protected by the security options. This means that addressing information and headers of higher layers are protected. In contrast, TLS can not encrypt the information in the IP-header since this information is necessary for the message to be delivered correctly by each intermediate device (i.e. router). Routers must know where to forward the packet and rely on the destination IP address to determine the next-hop. Another benefit is that integrity protection is possible for the entire data payload and for the MAC headers themselves. As a result, the addressing information for higher layers cannot be tampered with, i.e., it has integrity protection.

In contrast to layer 2 security solutions, IPsec (see Section 2.7) and the SMACK extension provide end-to-end security since they are based on the network or application layer. This means that the intermediate devices do not have to share keys or be involved in any way in order for the security mechanisms to function properly. This can be a great advantage when dealing with traffic over networks where there the end device owers/operators lack control, such as the Internet. In many scenarios, IoT devices will receive packets from a remote device over the Internet like, for example, when a user remotely reads some value or reconfigures an appliance at home. Furthermore, using an application layer mechanism means that many different types of networks can easily carry the traffic, for instance: IEEE 802.15.4 and IEEE 802.3 Ethernet links. End-to-end security also ensures that even if one or more devices in the network are compromised the security of the end-to-end traffic is not degraded, at least for well-designed security protocols.

2.6.2 Exploiting IEEE 802.15.4 encryption and authentication

In the interest of minimizing power consumption, it would be beneficial if the gateway implemented IEEE 802.15.4 encryption and authentication, thus reducing the need to do this at higher layers in the nodes. This would split the security into two parts, rather than providing end-to-end security.

Moreover, it could decrease both the information that needs to be sent over the wireless link and reduce the computations necessary at the constrained device. However, this solution is *outside* the scope of this thesis project, hence it will not be addressed here, but remains for future work.

2.7 IPsec

Internet Protocol Security (IPsec) is a popular protocol for securing IP traffic with regard to confidentiality (by employing encryption), data integrity, and origin authentication. Its main purpose is to protect data in IP packets by defining the steps and protocols to achieve this. The protocol was first standardized by the Internet Engineering Task Force (IETF) in 1998 in RFC 2401 [36] and further updated in RFC 4301 [37]. This protocol is based upon earlier research protocols*, such as swIPe [38]. Some of these protocols had overly complex specifications [39] and hence it was decided that there was a need for a standardized and secure protocol that would take into account the benefits and lessons learned from the existing options at the time.

One benefit of IPsec is that many different encryption algorithms are supported [37]. Furthermore, IPsec supports key management, session handling, replay protection, and more. IPsec defines a complete security infrastructure that can be used to deploy secure communication. IPsec is a well-tested protocol that is widely used to realize Virtual Private Networks (VPN) [40]. Since IPsec is implemented at the network layer for both IPv4 and IPv6†, it can support any higher layer protocols, such as TCP or UDP. The advantage is that no higher layer protocol needs to be customized to work with IPsec; instead every higher layer protocol can run transparently over an IPsec security association. This is a strong point since it reduces the work needed for adding security to a network. Neither the devices themselves nor intermediary systems need any major modifications to enable IPsec.

Some of the IPsec disadvantages include the fact that it is a quite complex system with many parts. The protocol is dynamic and can support a large number of configurable settings. Unfortunately, this large number of settings makes a complete implementation more difficult to create. The packet overhead for transmitting data is in the order of 50-80 bytes [42]. Performing the encryption and authentication steps also requires processing power. The amount of processing power depends on the chosen algorithm. Because of this IPsec may be impossible to implement on devices that are too constrained in terms of processing capacity or devices with severe limits on available electrical power [43]. The bandwidth overhead can also be a problem in low bandwidth networks, especially when small packets are frequently sent, thus making the overhead a significant part of the total data sent.

2.8 Secure Real-time Protocol

The Secure Real-time Protocol (SRTP) [44] addresses the case where there is a series of small amounts of data that need to be transmitted securely. It supports confidentiality, authentication (optionally), and replay detection - while adding only four bytes to the size of a Real-Time Protocol (RTP) [45] message. It does this by *taking advantage* of the RTP packet already including a sequence number and timestamp. Note that SRTP can tolerate packet loss. The protocol uses AES for encryption and a Hash-based Message Authentication Code (HMAC) based on the SHA1 hash function. Data confidentiality is realized by replacing the original RTP payload with an encrypted version. As for basing the HMAC on SHA1, even if some collisions or other security issues are found with SHA1, as is the case with MD5, this does not necessarily mean that an HMAC based on SHA1 will be compromised [46].

As mentioned above, the overhead compared to normal RTP traffic is very low. The only new fields defined are an optional field that identifies the master key used and a recommended field with authentication information. Fortunately, RTP already supports functionality typically needed for replay detection and mitigation of other common security flaws in the form of sequence numbers and timestamps. SRTP does *not* provide confidentiality to the RTP packet headers, the reason for this is to

* A list of some of this research can be found in the survey: http://web.mit.edu/tytso/www/ipsec/surv9703.html
† Current standards specify that IPsec support **should** be implemented in any IPv6 nodes [41].

allow header compression. If there is need to secure the packet headers, then the SRTP RFC recommends using another protocol, such as IPsec.

Garg, Singh, and Tsai analyzed the security of SRTP and note that due to the use of HMAC-SHA1 the protocol is susceptible to DoS attacks [47]. Because the HMAC calculation incurs overhead, flooding the receiver with SRTP packets can overwhelm it. The authors propose two different schemes to solve this problem. These two schemes combined are called SRTP+ and both are based upon the idea of adding another level of authentication that is cheaper to calculate. If a device is flooded with SRTP packets with incorrect HMAC values, the receiver utilizes a simpler protection method to quickly discard invalid packets. This additional layer imposes only a small overhead for legitimate traffic, but can avoid unnecessary processing in the case of DoS attacks. In comparison SMACK only needs to perform HMAC calculations for approximately every 16^{th} packet (using the default configuration) rather than for every packet sent/received.

SRTP+ Scheme 1 uses a shared seed for a pseudo-random number generator (PRNG) to provide authentication, both devices will generate the same values from the PRNG and thereby are able to confirm if a packet is authentic. For instance, the sender will transmit a message with the 10^{th} output from the PRNG sequence as authentication and the receiver can confirm this by checking that the 10^{th} value of its PRNG gives the same result. Since both of them use the same starting seed, the results will match if the message is authentic. The seeds have to be exchanged in a secure manner during a setup phase, before data transmission starts.

SRTP+ Scheme 2 is even simpler and uses pre-computed numbers for authentication. The authentication values for the next N packets are periodically provided to the receiver. These value are encrypted and transmitted as part of a SRTP payload. After both parties share these same random numbers they are used as a one time key for each packet. The receiver checks if the incoming packet contains the next expected number and if so it accepts the packet. Each number is only used for one packet so the sender needs to keep supplying these numbers leading to a small increase in communication overhead. Test results show a speed improvement of at least 3.5 times for scheme 1 and 8 times for scheme 2 in comparison to not using either of these schemes.

2.9 Multimedia Internet KEYing (MIKEY)

A common question for a security protocol is how to distribute or generate keys. The Multimedia Internet KEYing (MIKEY) [48] protocol is used to provide SRTP with session keys. One of its stated goals is to provide a key management system with end-to-end security. Other goals are simplicity, efficiency in terms of overhead and independence from the underlying protocols. A popular method for key distribution and management is the Internet Key Exchange (IKE) protocol (used by IPsec). However, as the MIKEY RFC [48] states, streaming data has special needs and needs a protocol better adapted to it. MIKEY is primarily intended for use with simple peer-to-peer connections or groups of small size. The system also supports a variety of scenarios, such as unicast, multicast, and many-to-many communication. In contrast, IKE does not support multicast scenarios in a reasonable manner since each security association is between *pairs* of devices. In order to support multicast communication if n is the number of devices, then $\frac{n(n-1)}{2}$ security associations will be necessary when using IKE.

Another key point is that it should be possible to integrate MIKEY data in other protocols to avoid having to do MIKEY negotiation separately. Thus MIKEY should be included in the session establishment of other protocols as much as possible. How this can be accomplished is described in RFC 4567 where key management protocol support for Real Time Streaming Protocol (RTSP) and Session Description Protocol (SDP) among others are described [49]. RFC 4567 provides a framework describing how key management protocols can interact and carry their messages in RTSP or SDP traffic. Both RTSP and SDP are extended with new headers that support the required options for key management. Some important options added are an identifier that specifies the key management protocol used and a data field where whatever data the key management protocol wants to relay is placed. An important requirement of the key management protocol is that the initial step of the protocol must be possible to perform in a single request-response message exchange. MIKEY is

specifically mentioned in RFC 4567 and example scenarios where it is used are provided. MIKEY supports the previously mentioned requirement of needing few messages to initialize the shared keys. Security-wise a potential problem highlighted is that some protocols such as Session Initiation Protocol (SIP) utilize intermediate proxies. This can prevent the session setup traffic from being secured end-to-end. One implication of this is that an intermediary can intercept keys and use those to attack the encryption of the media stream created by the media delivery protocols. One solution to this is of course to use Secure/Multipurpose Internet Mail Extensions (S/MIME) to secure the SDP so that the proxies cannot see the MIKEY information.

MIKEY uses AES in counter mode for encrypting the keys to be delivered, while authentication is provided by a HMAC based on SHA1. To create a secure communication session for distributing keys three methods are described: pre-shared keys, Diffie-Hellman key-exchange, and public-key encryption. For speed and efficiency the RFC recommends pre-shared keys, but notes that for larger systems this can be problematic. Public-key cryptography is more scalable, but requires a Public Key Infrastructure (PKI) to work optimally; in addition it is more resource consuming as symmetric encryption is faster than asymmetric. Diffie-Hellman key-exchange is more resource intensive than the previously mentioned methods and also requires PKI systems to ensure user authenticity and protect against man-in-the-middle (MITM) attacks. Several later RFCs add additional support for new methods to create a session and negotiate a common secret in addition to the three mentioned above.

While the purpose of MIKEY is to distribute keys to systems, it still needs key information to be present in those systems, with the exception of Diffie-Hellman key-exchange where the end nodes generate the keys. The actual keys that MIKEY distributes to systems can be calculated either from pre-shared keys or a shared piece of data agreed upon by the devices during the initial MIKEY messages. Derivation of these keys is done with an HMAC based on SHA1. Timestamps are used to provide replay protection, which means that the clocks have to be synchronized. Clock synchronization is also used to reduce power consumption when using IEEE 802.15.4 and for WLANs operating in infrastructure mode the nodes also synchronize their clocks with the AP. To protect against replayed messages within the acceptable time window, a replay cache keeps track of the accepted messages that arrive in this window. A replay cache of 6 kB is assumed to be reasonable for most cases. This size is large for constrained devices as they may only have 10-50 kB of RAM. For extreme cases, a cache of up to 48 kB is mentioned in the MIKEY RFC [48] p. 31. In contrast SMACK requires only ~16 bytes for replay protection (using the default session length) as will be described in Section 3.6.

2.10 DTLS

Datagram Transport Layer Security (DTLS) is a protocol for encrypting UDP traffic based on the Transport Layer Security (TLS) [50] protocol, TLS is used for encryption of HTTP traffic, among other uses. DTLS was first presented in 2006 in RFC 4347 [51] and later updated in RFC 6347 [52]. In these RFCs, DTLS is presented as a series of deltas, specifying how and when it differs in implementation from TLS. The purpose of the protocol is to provide the same level of security that TLS provides to TCP traffic, but applied to UDP. Some of the common protocols used on the Internet such as Domain Name System (DNS) and many systems for Voice Over IP (VoIP) communication utilize UDP as their transport layer protocol. DNS has in the past been targeted by many attacks since it is a high value target. These attacks attempt to redirect users to fake websites by providing forged DNS entries. TLS does not support encrypting UDP traffic and because UDP cannot use the standard implementation of TLS, it therefore requires another method to achieve confidentiality and secure the user’s communication.

There are some key differences between how DTLS and TLS functions. One main difference is that UDP does not have any built in functionality to ensure that packets are delivered to an application on the receiver in the correct order (as UDP lacks any concept of byte stream ordering). TCP uses sequence numbers and a request/acknowledgement scheme to ensure that bytes are reliably delivered and ordered correctly for delivery to the application layer, while UDP lacks this functionality [53]. This means that DTLS has to implement this functionality on top of UDP and do so at the application layer rather than at a lower layer in the TCP/IP stack. This is logical since DTLS must be self-

contained and function without requiring any modifications to the lower layers. By reimplementing some of the functionality of TCP in DTLS the necessary benefits of TCP can be transferred to DTLS even though it is running over UDP.

DTLS adds support for numbering and functionality to mitigate packet loss [51]. This is accomplished with sequence numbers in a similar fashion to TCP. DTLS also adds support for automatic packet retransmission, reordering, and replay detection. By combining all these features DTLS accomplishes what TCP and TLS together achieve. Some other considerations are that stream ciphers such as RC4 cannot be used with DTLS since they rely on the ordering of the data and make packets interdependent on each other, because of this stream ciphers are banned in DTLS [51] simply because the protocol would not function when using them.

A potential security issue that has to be taken into account is DoS attacks, which can render the attacked device unresponsive. Because the source IP address of the device making a request is not verified an attacker can spoof messages and consume memory resources of the receiver by setting up fake DTLS sessions. To protect against this DTLS uses a concept called stateless cookies. These cookies force the sender to prove that it can both send and receive data on the IP address it is using [51], this greatly increases the difficulty of spoofing the source IP address. Apart from these changes, DTLS is very similar to TLS and this is a strong point since TLS is one of the most widely used security protocols on the modern Internet. This is also one of the main reasons DTLS is presented as deltas compared to the full scale TLS protocol, only parts of the implementation necessary to adapt the protocol to UDP need to be changed. As the TLS protocol itself is secure [54], as many parts as possible should be left unchanged.

For UDP based traffic, such as CoAP, DTLS is a potential choice for protecting the traffic. However, there are issues with expensive cryptographic operations that have to be performed. This is especially important when the DTLS protocol is executing on constrained devices. A thesis by Stefan Jucker [55] explores the drawbacks and benefits of using DTLS with CoAP with a focus on the Californium library. Stefan Jucker found that DTLS is currently *unsuitable* for constrained devices [55], because the implementation uses too much memory and processing power to be appropriate for constrained devices.

Another problem is the data overhead induced by using DTLS; since static length header fields are used the overhead can be significant. The most expensive parts of DTLS's operation is session establishment. Running CoAP over DTLS can induce a delay of 40-130 ms and an overhead of 29 bytes [55]. Additionally DTLS requires more messages to be sent to start a communication session than CoAP does. Establishing a session beforehand and reusing it shows much better results with a resulting delay of only 5 ms. However, because devices often go to sleep and communicate with many other devices simultaneously the DTLS handshake will have to be performed frequently. Especially in the case of sensor networks, one node can have many neighbors that it needs to communicate with simultaneously. It is noteworthy that some protocols such as MIKEY and SRTP (see Section 2.8) avoid this problem. In their case, the only additional cost is for the initial MIKEY key exchange and that can be done in one round trip plus the time for some local processing.

2.11 Lithe: Lightweight Secure CoAP for the Internet of Things

Lithe [56] proposes DTLS header compression for use with CoAP. Because DTLS was originally designed for reliable links with high bandwidth it is not ideal for constrained devices. DTLS introduces some overhead for each packet that it protects. On constrained networks, this extra overhead leads to additional radio usage. Lithe attempts to alleviate this problem by creating an integrated DTLS and CoAP system for the IoT. The goal of this solution is to reduce power consumption, while maintaining the end-to-end protection DTLS provides, through reduced packet sizes.

Header compression for 6LoWPAN can compress the IPv6 headers and the UDP headers, while correctly dealing with the source/destination ports and checksum [32]. Lithe extends this functionality to the UDP payload by defining a new encoding type that allows the protocol to signal that the UDP

payload itself is also compressed. The UDP payload is assumed to be DTLS traffic and the targets of compression are the DTLS headers. The DTLS message types that have compression rules defined are Handshake, Record, ServerHello, and ClientHello messages. Some types are left uncompressed, as no fields suitable for compression are available in them, as is the case with the ServerHelloDone, ClientKeyExchange, and Finish messages. Fields that are important to maintain security such as the random-field that contains random data used for encryption purposes are left uncompressed and unaltered.

Tests by Raza, et al. show a large decrease in overhead ranging from 14-100% depending on the message type. To achieve 100% savings Raza, et al. assume that some pre-shared information concerning certificate types, certificate authorities, and algorithms are available to the devices on the 6LoWPAN network. This allows them to omit all fields in the CertificateRequest message. In addition to reduced overhead, the size of the implementation is small, requiring only 59.4 kB of ROM and 9.2 kB RAM. Energy consumption is also reduced when using Lithe. As to round trip time, Lithe takes slightly longer than CoAP with DTLS in most scenarios. This shows that it is possible to reduce the overhead when using DTLS with CoAP in a power efficient manner *without* greatly increasing the round trip time of packets on the network.

One specific problem mentioned by Raza, et al. is that if 6LoWPAN is forced to fragment a message due to its size the round trip time is greatly increased. This effect can only be seen when using CoAP in combination with DTLS, as this does not happen when using CoAP with compression enabled or CoAP alone. One of the design goals of the CoAP protocol is precisely to avoid fragmentation as much as possible [12]. However, when DTLS is enabled the extra overhead added by the DTLS header information can cause fragmentation. Lithe solves this by compressing enough of the DTLS headers that fragmenting packets can be avoided to a great extent, specifically 64 bytes of extra payload is available before a packet has to be fragmented compared to uncompressed DTLS. Lithe also saves power by reducing radio communication since a packet that is fragmented in two transmissions will utilized the radio more compared to transmitting the same packet unfragmented.

2.12 Analysis of Existing Internet Protocols for the Internet of Things

In 2011, Heer, et al. did an analysis of existing Internet protocols and their applicability to IoT [57]. In their paper they consider limitations of traditional Internet protocols and what special challenges arise for IoT. The following paragraphs cover some of the challenges with regards to security that they identified for IoT.

One major issue and a defining characteristic of IoT is that both the network itself and the devices have very limited resources in terms of bandwidth, memory, Central Processing Unit (CPU) capabilities, and available electrical power. Because of this some technologies such as public key encryption, which is very resources intensive, are less suited to the IoT. Furthermore, the small link MTU size before fragmentation of packets occurs introduces the possibility of attacks and performance loss due to fragmentation. Their paper also notes that assumptions cannot be made about the power usage of a specific protocol *unless* an implementation is actually made for specific IoT devices. Because of the limited resources, the susceptibility of IoT devices to DoS attacks is heightened. When resources are more limited, exhausting them is easier and occurs more quickly than for conventional mains power computer systems. The main targets of exhaustion are battery power and RAM. Protocols such as IKEv2 and DTLS avoid creating state for a connection *until* the address of the other party has been verified. This can protect against DoS attacks when an attacker uses a spoofed IP address as the source IP in an attack. By not creating state until the connection has been verified makes the process of creating countless spurious connections made more difficult -- as it puts added constraints on the source of the connection.

Another issue is that interconnecting the IoT with the Internet can interfere with end-to-end security. When security protocols protect header information of packets, then these headers cannot

easily be rewritten or modified by gateways (when needed)[*]. One proposed solution is to share keys with the gateway, however this weakens the system's security. Another option is to use the same packet format on the IoT and the Internet, thus avoiding the need for rewriting packets - although this can reduce performance in the IoT. A third option is to only protect specific parts of a packet and leave other parts that can be modified, thus an appropriate tradeoff between security and performance is important. Finally, the last alternative mentioned is to use advanced MACs that allow for some transformation of messages without breaking integrity, but this solution is more complex and difficult to use for encrypted data.

Key distribution and defining identities for each device is another challenge. For instance, one way this can be done is in a distributed way is for devices to form *ad hoc* security associations and share keys as needed. Another option is to have a centralized system that distributes identity information and keys to devices, but a drawback of this is the introduction of a single point of failure. Distributing certificates and bootstrapping information can be more cumbersome in the case of constrained device and networks, as certificates and keys can be relatively large. Privacy issues should also be considered, some protocols such as DTLS allow the client to remain anonymous by requiring authentication only of the server. However, just as in the case of TLS authenticating only one party can lead to MITM attacks. Despite this there is also an advantage in allowing one-way authentication as the server-client relationship means that it is more common to have a trusted server and unknown clients that must be authenticated to gain access.

In conclusion, Heer, et al. emphasize that solutions should scale from small to large scale networks. Additionally, they note that is important to consider not only end-to-end security solutions, but also consider systems that will work well when securing communication for larger groups. Which layer to secure in the IoT remains important for researchers to consider as there are advantages to placing security at each layer of the network stack, but resource limitations make it difficult to secure all of the protocol layers. One specific concept that protocols working in the IoT should take into account is the need for providing security and sharing keys between layers.

[*] This is particularly an issue when using network address translation when using IPv4 addresses for IoT devices.

3 SMACK

The Short Message Authentication ChecK (SMACK) is a proposed security extension to CoAP. This protocol adds a method for lightweight authentication of messages to CoAP. Its main goal is protection against battery exhaustion and denial of sleep attacks. Currently a proof of concept implementation exists written in Java by Marco Tiloca at SICS [58]. The specification of the SMACK extension will be used to create a C version that extends the Erbium CoAP implementation on Contiki. SMACK requires some modifications to function well on constrained devices. Specifically, the memory footprint and processing power required should be reduced. SMACK also has to be adapted to fit with the REST model that Contiki uses to implement protocols such as CoAP.

SMACK is an attempt to create a robust and lightweight authentication extension to CoAP. The current Java prototype implementation of SMACK is written on top of the Californium library. The current implementation functions on full feature devices, but needs to be adapted and implemented in C to run on constrained devices. Few of these constrained devices can run the Java runtime and execute Java programs, thus most constrained device require an implementation that uses a language operating closer to hardware, such as C.

Technical details of the SMACK extension will be covered in later sections of this chapter. Briefly, SMACK relies on using a MAC to authenticate messages. This MAC acts a signature that is attached to each message sent, so that the receiver can verify that a given message is correctly authenticated and thus should be further processed Locally computing a matching MAC can be considered proof that the sender and receiver share some secret data (such as encryption keys) [59]. . A MAC is typically lightweight to compute and small in size. Calculating this MAC should be secure **and** resource efficient.

3.1 Overview

The main result of the SMACK extension to CoAP is to introduce a MAC in a section of the token field. The token field is specified in the CoAP header to differentiate between different communication sessions. The length of this field is variable and between 0-8 bytes. SMACK takes advantage of this field and the fact that it is already defined in the standard. This means that no new fields need to be defined and the necessary modifications to the protocol are small. In place of the token field SMACK introduces two subfields, one that serves the same purpose as the old token field named "request ID" and another field named "validity check" holds a MAC. By default, a 4 byte long token field is used which SMACK subdivides into a 2 byte Request ID subfield and a 2 byte Validity check subfield.

Another advantage of reusing the token field is that SMACK is backwards compatible with CoAP devices that are not using this field. If a SMACK request is sent to a server that does not implement SMACK this server will place a copy of the received token in the outgoing packet and reply with that. This is the standard operating procedure for CoAP, using the same token used in the reply as was used for the request. Since the MAC is a part of the token field this overloading of the token field is completely transparent to devices that do not use the token field or implement the SMACK extension. However, devices that implement SMACK can differentiate between the sub-fields of the token field and can check the MAC. Splitting and reusing the token field in this way causes no additional problems since this field was optional from the beginning. Additionally, SMACK retains the same functionality the token field provides, but reduces the number of bytes that can be used for tokens by 2 bytes as the MAC uses 2 of the 8 bytes available in the token field.

The goal of SMACK is to ensure protection against DoS, specifically denial of sleep and battery exhaustion attacks - particularly for constrained devices. Protection against these types of attacks is important since most constrained devices have a very limited source of power. For example, if the device is battery powered then an attacker can drain the battery by sending request messages and thereby causing the radio and processor to use up all of the available battery power. In many cases, once the battery is drained it may never be replaced or replacing it can take a lot of time and effort. Sensor nodes in particular can be spread over an area and an individual node may never receive service or replacement of faulty parts. Additionally, nodes may be placed in difficult to reach places such that

sending someone to replace the battery is infeasible or simply not worth the cost compare to the benefit gained.

In a scenario without SMACK an attacker may send hundreds of request messages, eliciting a reply from the receiver. In addition to the processing power needed to deal with parsing the request and creating a reply (which may contain sensor data or other data that takes power to produce), an attacker can request a reading from a specific sensor or request a particularly large CoAP page. In contrast, when using SMACK messages are authenticated and those that fail the authentication check are discarded before any further processing and unnecessary radio traffic has occurred, thereby preserving battery power. Unfortunately, the radio has to be on in order to receive the message and a minimal amount of parsing has to be done in order to check the MAC, but if the MAC is incorrect the message can thereafter immediately be ignored – hence avoiding the rest of the processing and potentially enabling the radio to be placed into a sleep mode sooner.

The purpose of the MAC in the header is to enable message authentication, thus allowing a receiver to check whether a received message is authenticated and therefore should be accepted. The protection provided by SMACK and the MAC is a lightweight message authentication of parts of the message. Only certain elements of the CoAP header are authenticated, as will be detailed in coming sections. The MAC must be quick to calculate, while still providing a reasonable level of protection. Selecting which specific parts of a CoAP request should be included in the MAC calculation can be changed if some parts of this information are more important to protect.

3.2 Keys

There are several layers of keys used in SMACK. In a real world scenario, devices would receive initial keys from a key distribution center (KDC) (or via some other method of key distribution). These initial keys are called master keys and are used to generate the keys used for the MAC calculation. Figure 3-1 shows the relationship between the different keys used by SMACK.

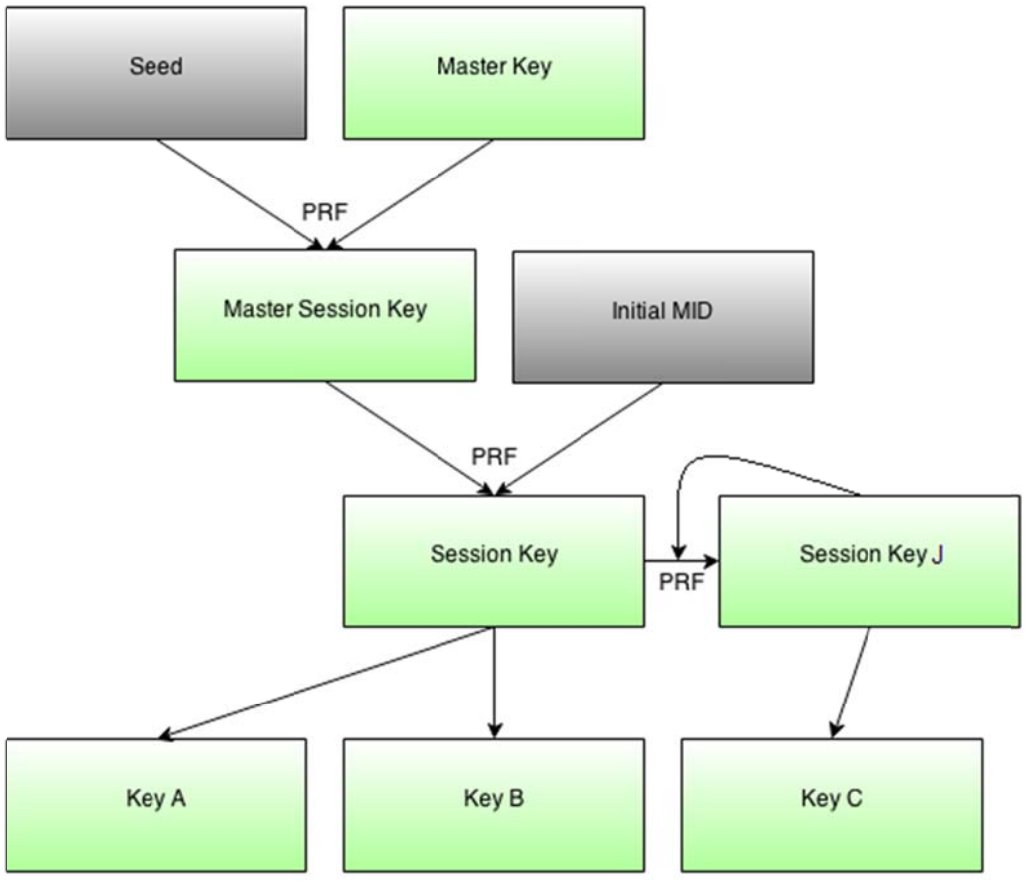

Figure 3-1: Keys used by SMACK

The KDC distributes the initial seed and master key to all devices involved in a communication session. The master key is now fixed and can be used to generate further keys. The master key and the seed are used as input to a pseudo random function (PRF) to create a master session key that is valid for a particular global session. This master session key is subsequently used together with the initial message ID (MID) of a CoAP packet to generate a session key for a specific session. A session is identified by the initial MID of the first packet received. For CoAP this MID is a 16-bit value in the header that is transmitted and automatically incremented for each message sent.

Furthermore, another instance of the session key is created in the form of Session key_J. The purpose of this new session key is to provide greater variation in the keying material. All keys except A, B, and C are 32 bytes in length by default. Finally, keys A, B, and C are generated as shown in Table 3-1. These keys are used by SMACK to generate the MAC to be written in the CoAP header.

Table 3-1: Generation of Keys A, B, C

Key	Size (byte)	Bit range (big endian)	Source
A	2	0 to 15	Session Key
B	2	16 to 31	Session Key
C	2	Start: $16 \times ((i+2)\ mod\ 16$ Stop: $16 \times ((i+2)\ mod\ 16) + 15$	Session Key_J $i = MID.J = \frac{i+2}{16}$

Keys A and B are generated in a straight forward fashion by simply taking a fixed segment of the Session Key. However, key C is generated from the constantly changing Session Key_J. The choice of which particular Session Key_J and which parts of it are to be used is determined by the message ID of the CoAP packet in question. For instance if the packet has MID 53, then the Session Key_J used is Session Key number 3 and bits 112-127 of Session Key_3 are used to create key C. Rotating the Session Key_J and selecting different parts of it increases security because the same key is not used more than one packet. In this way, a new key C will be used for each packet. The cycle time of Session Key_J depends on the output from the PRF, if there is a case where continuously taking the initial Session Key plus the current Session Key_J as input to the PRF at some point loops, the same Session Key_J will be generated.

3.3 Pseudo Random Function

The keys themselves rely heavily on a PRF to generate good key material. A good PRF will generate statistically random data no matter what input material is provided [60]. This means that a small change in the input material to a PRF will result in vastly different output. Ideally there should be no discernable relationship whatsoever between the input and output of a PRF.

There are different ways to implement a PRF; the method chosen for SMACK is to use the SHA256 cryptographic hash function as its base. Hash functions provide fixed length output values calculated from the variable data they operate on. Typically, hashes are used to verify the integrity of data or as means of storing passwords. The benefit of hash functions is that they are very difficult to invert, meaning that if the hash is provided, then finding the original input data is hard. In practice, attacking hash functions is typically based upon testing variations of the input data until the desired hash is found. When the input is small, for example a short poorly selected password, then the original data can often be found from a hash. SHA256 is a hash algorithm created by the United States of America's National Security Agency (NSA). It provides an output digest of 256 bits. Many other hash functions such as Message Digest 5 (MD5) have known security vulnerabilities [61]. Currently, SHA256 is considered a more secure hash function than MD5.

The HMAC chosen by SMACK is the same as implemented by the TLS protocol [50]. This HMAC is also used in other protocols, such as IPsec and DTLS. The HMAC specification is given in RFC 2104 [46] and is a commonly used and standardized mechanism for message authentication. The

original HMAC was first described in a 1996 paper by Bellare, Canetti, and Krawczyk [62] who also authored RFC 2104.

The SHA256 hash is used to implement a HMAC. A HMAC is a way to adapt hash functions to provide cryptographic security. In essence, a specific piece of data is hashed together with a key in several steps in order to increase the effort needed to reverse the function. The benefit of a HMAC is that data can be authenticated by using a specific key as one of the inputs to the function. For a normal hash function the output is always the same for a specific input, however since a HMAC uses the key as an additional input the output depends upon both the data and the key.

An HMAC can be used to check if some portion of the data has been modified or not. If the parties share a key, then they calculate the HMAC of the data with this key both before transmission and after reception. If the HMAC of the received message matches the expected value then the data has not been modified and can be considered authentic. If the HMAC differs from the expected value, then the data (or key) must have changed. An attacker cannot easily modify the data and recalculate the HMAC, as would be the case if a simple hash function was used, because the correct key is needed to generate a valid HMAC. If the key is well chosen, then an HMAC is a strong method for authenticating data. For this implementation of SMACK, it is up to the user to select a good Master Key that the rest of the keys used will be generated from. This key can be distributed by a KDC or preprogrammed into the devices. As the master key is 32 bytes long it will be difficult to recover this key -- assuming the choice of key is sufficiently random.

SMACK implements a PRF using a HMAC based on the SHA256 hash function. This PRF takes two values as input: a secret key and some arbitrary data to generate some output data. The main purpose of the PRF is as a wrapper to the HMAC to allow outputs of arbitrary length. To accomplish this it simply uses the HMAC multiple times according to the desired output length. Since SMACK by default uses a key length of 256 bits, the PRF only has to execute the HMAC once as the output from the HMAC is the same length as the hash function being used (and SHA256 has a 256 bit output). The PRF used by SMACK is similar in functionality to the PRF used by TLS.

The PRF is used to generate sub-keys derived from the main master key. For every new session, a new key is generated using the PRF from the master session key and the initial message ID of this particular session, as provided by a KDC or for the implementation described in this thesis the KDC is emulated in software. Furthermore, the Session Key_J is continuously refreshed by executing the PRF with the Session Key and last used Session Key_J as input. In this way the future values of Session Key_J rely on its previous iterations. Because the MID will wrap around to zero after 2^{16} messages it is necessary to also rotate and change the Master Key after 65 536 messages have been exchanged between the devices. How this is best done is an open question, but it could be done by using a KDC or other methods of key distribution.

3.4 Configuration values

Most values used by SMACK are possibly to modify and dynamically change to provide adaptability for different situations and requirements. For example, some hardware can have lower processing power or less memory available and some networks can have special characteristics. There can also be different security requirements and tradeoffs. Table 3-2 shows the default settings that SMACK uses for some key values.

The default size of the Token field in the CoAP header is 4 bytes, of these 2 bytes are the Validity Check (MAC) and 2 bytes are for the Request ID (i.e., the same purpose as the original Token field). The default key size was chosen to be 32 bytes to function smoothly with the hash and HMAC functions used that generate 32 byte outputs (thus the HMAC is used only once to generate all 32 bytes).

Table 3-2: SMACK key values

Name	Value	Description
SMACK_AUTH_FIELD_SIZE	4	Size in bytes of Token field in CoAP header
SMACK_VALIDITY_FIELD_SIZE	2	Size in bytes of Validity Check subfield
SMACK_GALOIS_FIELD_SIZE	16	Field size in **bits** used for Galois calculations
SMACK_KEY_SIZE	32	Key size in bytes of SMACK in bytes (256 bits)
SMACK_PORTION_SIZE	16	Controls how often Session Key$_J$ recalculates (in this case every 16th packet)
SMACK_SESSION_LENGTH	127	Length of a SMACK session (packets)
SMACK_ACCEPTANCE_WINDOW_SIZE	50	Upper limit for new session initial MID (multiplicative factor for the session size)

3.5 Galois fields

Galois field mathematics is used to calculate the value placed in the Validity Check subfield. Galois fields, also called finite fields, are defined as sets of numbers in which mathematical operations on the members of the set results in another member of the set [63]. SMACK uses a field size of 16, the range of such a Galois field is 0 .. 2^{16}-1 (i.e., 0 .. 65 535). For example, when using a Galois field of 16 bits the following calculations hold true: $260 \times 260 = 4123$ and $60000 + 20000 = 42048$ as the results of the calculations are also members of the set and remain within the range of the field.

Simple addition in a Galois field is performed using the *exclusive or* logical function $a \oplus b = c$. Multiplication uses algorithms based on primitive polynomials. Each field size can have many potential primitive polynomials. In essence, a primitive polynomial is an irreducible polynomial, the equivalent to a prime number but for polynomials. Multiplication in a Galois field is performed modulo the primitive polynomial used for the specific field size. It is a more complex operation compared to simple addition.

Often Galois field multiplication is performed using pre-computed lookup tables as multiplication is quite costly processing wise [64]. Many of the available implementations of Galois field mathematics in code rely on dynamically generating lookup tables that are loaded into RAM to assist with speeding up calculations. Since SMACK is developed for constrained devices, it cannot fully utilize such lookup table functionality for speeding up the calculations due to memory constraints. Many constrained devices have very limited RAM available and cannot afford large data structures permanently being loaded into memory. An alternative approach is saving pre-computed tables to the FLASH memory of the device, although this can introduce latency and the size required can still be too large for constrained devices. For instance, the code implementing Galois field calculations relying on pre-computed tables in James S. Plank's library [65] requires at least 1 MB of space because the code that creates the tables is the following: *malloc(sizeof(int)*nw[w])* and *malloc(sizeof(int)*nw[w]*3)* where a field size of 16 gives nw[16] = 2^{16} and an integer uses 4 bytes of space.

SMACK pre-computes certain values and utilizes a simple lookup table for some operations. SMACK has support for Galois field sizes from 1 to 16 and the primitive polynomials for each are stored in a table. The size of this table is 36 bytes. Using Galois field mathematics the three keys A, B, and C are used in addition to parts of the CoAP message header (m_1, m_2, and m_3) that are included in the protection according to the following formula:

$$MAC = (m_0 + A \times m_1 + A^2 \times m_2) \times B + C$$

Note that the multiplications and additions in the above formula are performed in a Galois field and thus follow the special rules mentioned earlier. An alternative representation showing the mathematical operations more clearly is the following formula where dot represents multiplication modulo a primitive polynomial:

$$MAC = (m_0 \oplus A \cdot m_1 \oplus A \cdot A \cdot m_2) \cdot B \oplus C$$

In addition to keys A, B, and C in the formula sections of the packets are also used. The m_i values are sections of the CoAP packets as illustrated in Figure 3-2. The colored sections are treated as normal integer values for the purpose of the calculation, thus m_0 is the fields: Ver, T, TKL, and Code; m_1 is the MID; and m_2 is the Request ID. Note that this means that the Options and Payload fields are *not* protected.

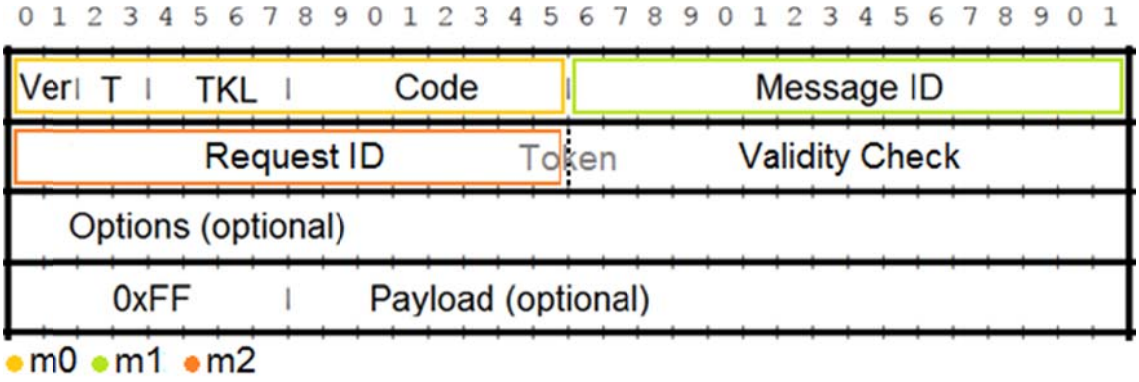

Figure 3-2: Message sections

It can be shown that the probability of accomplishing a forgery attack is 2^{-16} when using a field size of 16, as there are 2^{16} possible values for the MAC resulting from the computation. Thus it is unlikely to guess the same MAC as that of the next valid message the receiver is expecting. An adversary would have to transmit 2^{16} messages to ensure that one of them will match the MAC the receiver is expecting. Because of this an attacker that is capable of transmitting 2^{16} messages per second, having them all arrive and be processed by the receiver, will have their attack capacity reduced to $\frac{2^{16}}{2^{16}} = 1$ accepted message per second. This limits the efficiency of an attacker in terms of the number of messages the receiver has to fully parse. However, although the received messages are not fully parsed power is still required to power the radio and do the limited parsing required to calculate the MAC as can be seen in the experimental results from Chapter 5.3.

3.6 Replay detection

One well known attack against authentication systems is replay attacks [66]. A replay attack is executed when an attacker sniffs the network traffic, captures a correct message going to the receiver, and retransmits it later. The key here is that the MAC in the packet is valid and proves that the message is authentic. Since the packet is valid, the receiver will accept it *unless* there is some protection against replay attacks. In this way, an attacker can trick a system into accepting commands contained in previously sent valid packets. This attack is quite clever since it does not require any initial interaction with the device to be exploited; simply passively listening on the network is enough. Once the required packet has been captured, then the attack can commence, in many cases the packet will even appear to originate from the original sender since it is an exact duplicate. This can assist an attacker in hiding their identity.

In the case of SMACK, an attacker can capture a CoAP request with a valid MAC in the validity check subfield of the extended CoAP header and resend it later. This could allow an attacker to bypass the security SMACK provides and the attacker can send this specific packet again and again, requesting resources or some other action depending on what was in the original packet. There are no obvious distinguishing characteristics that the receiver can use to realize the packet is replayed, since it is identical to a valid packet. If such an attack occurs, then this causes problems for the receiver. As a result it is desirable to find methods that can prevent replay attacks.

SMACK implements replay protection in a simple and straight forward manner. First, when a message arrives SMACK checks whether it is a part of an existing session or not. This can be done by simply comparing the MID of the incoming request with the initial MID + session length for all *current* sessions. If the MID of the incoming request falls in the MID range of an existing session, then this message is accepted. If it is not part of an existing session, there are two cases. The MID is either evenly divisible by the session length, in this case a new session can be created, otherwise the message is discarded. The exact Initial MID that is assigned to a session can be provided by a KDC to ensure that both sender and receiver agree on only one allowed Initial MID.

For each individual session, a bit array is kept of the messages that have been received. With the default session length of 127, this array will be $\left\lceil \frac{127}{8} \right\rceil = 16$ bytes long. When a message arrives the corresponding MID in the array is marked as received. If a message with ID 439 comes in this will mark bit $439 \; mod \; 127 \; = \; 58$ in the bit array. The benefit of this solution is that the memory required is small and could be further reduced by reducing the session length. For a graphical representation of SMACK packet processing and replay protection see Figure 3-3. This figure clarifies the processing for several different potential scenarios.

One drawback of the method described so far is that a limited form of replay attack is possible. It is not possible to reuse a message for the same session (or for different sessions), however when the message ID loops around and starts over a message with an old message ID can be reused. For instance an attacker can capture a packet with message ID 200, wait 65 535 (2^{16}) messages and then retransmit it. The reason this is possible is that the maximum message ID is 65 535, after that value is reached the count restarts at 0. Now new sessions will be created again and the message IDs will not be marked as read. This limits an attacker to only replaying a message once until waiting for the message ID to start over again. This problem has been solved in the SMACK implementation by simply changing the master key every 65 535 messages. The burden of this is not too large since a large number of packets can be transmitted before having to change keys.

The following section will elaborate more on the functionality of the protocol in an example scenario.

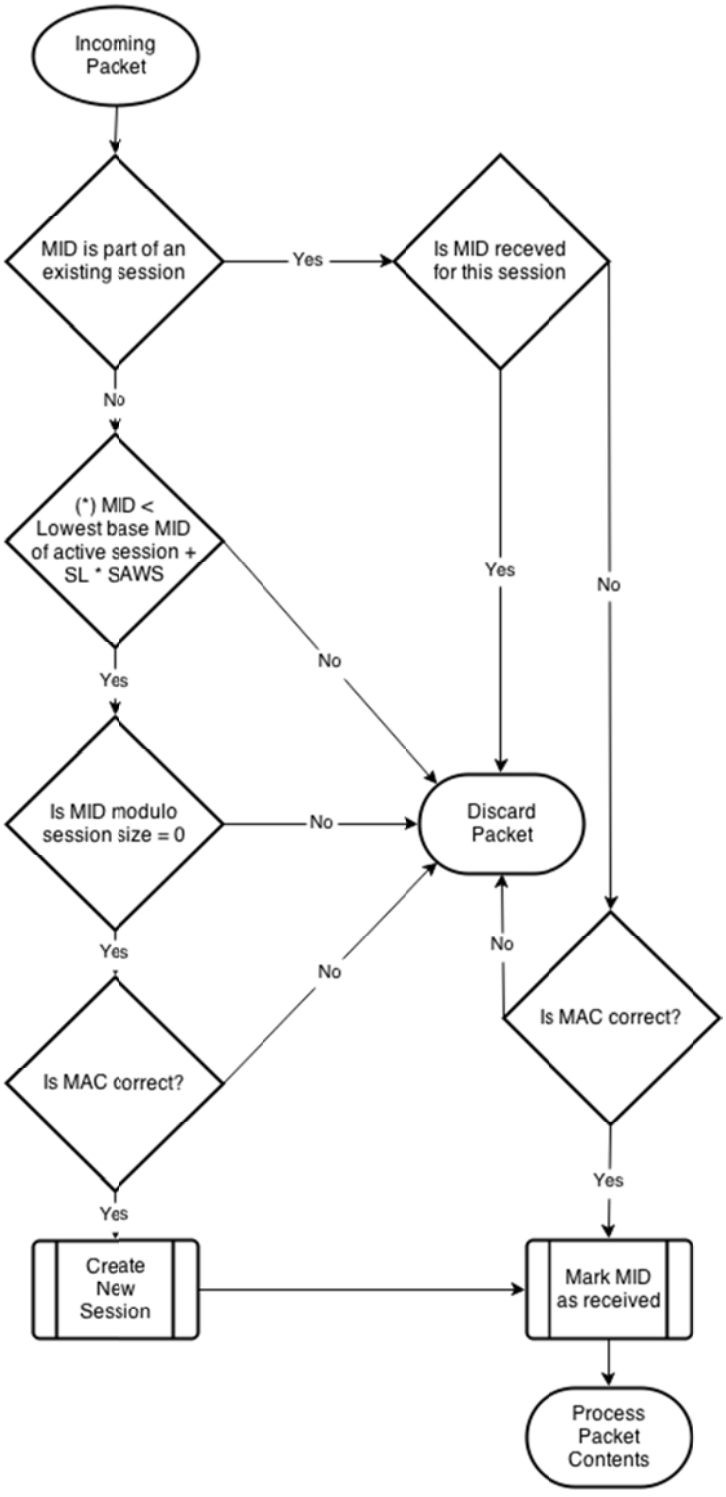

Figure 3-3: Packet processing flowchart: * To clarify this step every created session is stored along with its initial MID. The initial MID is the MID of the first packet in this session. By using the base MID + SMACK_SESSION_LENGTH it can easily be determined which session an incoming MID belongs to. A session will be deleted when all packets from the base MID to the base MID + SMACK_SESSION_LENGTH have been received since it means all packets in this session have been accounted for. The lowest initial MID of an existing session is the lower limit for acceptable MIDs and the upper limit is reached by adding the SMACK_ACCEPTANCE_WINDOW_SIZE (SAWS) * SMACK_SESSION_LENGTH (SL). This puts a range limit on acceptable MID values for new sessions and in addition to that new sessions must start on MIDs that fall on even multiplies of the session length.

3.7 Example scenario

This section and Figure 3-4 describe a possible setup and example of how the SMACK protocol will be used to extend CoAP to providie authentication for a client communicating with a server device. There are three devices in the network: a server, a client, and a KDC. The following steps prepare the devices and then exchange authenticated messages from the client to the server:

1. The client requests a Master Session Key and an Initial MID from the KDC.
2. The client receives Master Session Key and an Initial MID from the KDC.
3. The client creates the Session Key and Session Key_J from the Master Session Key and Initial MID using the PRF. Then it generates keys A, B, and C for the first packet. For future packets the key C generated depends on the amount of packets transmitted within this session since Session Key_J and the section of it used for key C depends on the MID of a packet.
4. The client uses keys A, B, and C to calculate a MAC for the first packet to be sent. This MAC is placed in the last 2 bytes of the Token field of the CoAP header. The first 2 bytes are filled with a random value to provide the original functionality of the Token field (i.e., to identify a sequence of messages).
5. The client transmits the first CoAP packet protected with SMACK.
6. The server requests a Master Session Key and an Initial MID from the KDC.
7. The server receives a Master Session Key and an Initial MID from the KDC.
8. The server receives a packet and first checks if it matches the Initial MID received from the KDC. If so the server generates Session Key, Session Key_J, and the three keys A, B, & C.
9. The server then uses the same algorithm to calculate a MAC for the packet and checks if it matches the one included in the packet. If it does a session is created and the packet is accepted and marked as received, if not it is discarded.
10. The server replies to the message. SMACK can function either as one-way or two-way authentication meaning the server can choose to embed a MAC or not.
11. Since the client has incremented the amount of transmitted packets key C is recalculated from Session Key_J and if needed a new Session Key_J is created from the Session Key and the current Session Key_J using the PRF. That will happen every time (i + 2) / 16 is incremented by a whole digit, meaning every 16^{th} packet.
12. The client now uses the keys A, B, and C to create a MAC for the second packet to be sent.
13. The client transmits the packet.
14. The server receives the packet and calculates the Initial MID for this session from the incoming packet's MID. This can be done by taking the MID – (MID mod SMACK_SESSION_LENGTH). Then the server checks if there is an active session matching that Initial MID, if not the packet is discarded. Next the server checks that this MID has not been previously received (by checking the bitmap of received packets), if it has it is discarded. If it has not been previously received the MAC is checked by first recalculating key C (and Session Key_J if needed) and then using the three keys A, B, & C to check the MAC.

    Whenever all the packets with MID starting from base MID up to base MID + SMACK_SESSION_LENGTH have been received a session is deleted. If the incoming MID of a packet is more than SMACK_ACCEPTANCE_WINDOW_SIZE * SMACK_SESSION_LENGTH over the lowest base MID of an active session the packet is rejected.

    New sessions will be created by the server when packets with MID that fall on Initial MID + SMACK_SESSION_SIZE * n are received assuming they are not duplicates and the session does not already exist. In a real scenario the KDC may be contacted to get new values for

Initial MID and keys if needed, however the current implementation simulates this. The value of the Initial MID is used to create new session and also to identify it.

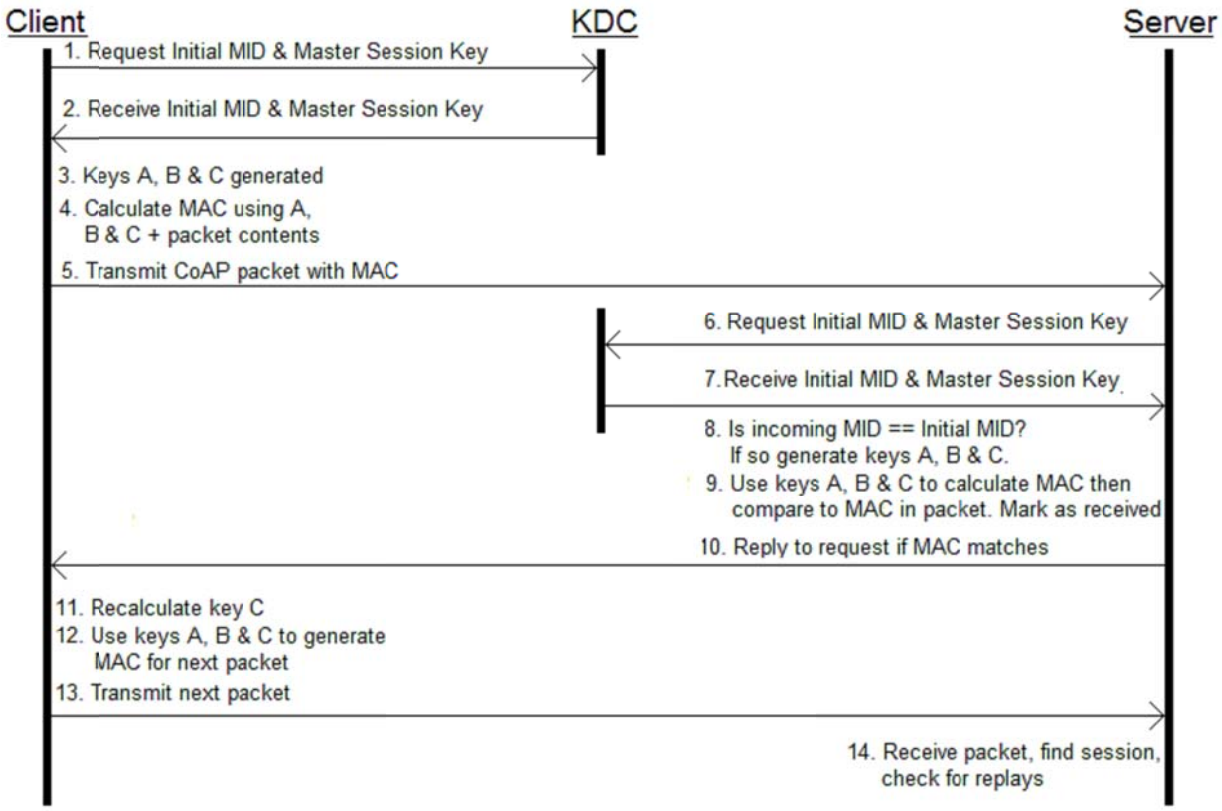

Figure 3-4: Communication steps

4 Method

The practical work performed as part of this thesis project extends the code of the CoAP implementation in Contiki, named Erbium, to add support for the SMACK extension. The current version of Contiki included with the Instant Contiki VMware image has an implementation of CoAP that supports the CoAP version 13 draft. Because this version was the latest available with the Contiki development environment at the start of the project, this version was used for the SMACK implementation. Erbium is written in C and can be used by Contiki applications if they wish to communicate using CoAP. This C code was modified to provide SMACK functionality for new and existing CoAP implementations. As mentioned in Section 2.5.2, the Cooja simulator and other tools are available to facilitate developing software on Contiki. This simplified the code development. Additionally, Cooja was used as a simulator to test code before deploying it.

The software development method used was incremental development, thus the SMACK functionality was systematically added to the existing system. First, the MAC calculation was included in the client code. Next the server was modified to check the MAC before accepting a message. The client could be fully developed before creating the server, as a SMACK client can interact with a non-SMACK server. The existing Java implementation was used as a baseline for creating the C version. Furthermore, Marco Tiloca at SICS who developed the Java version, was available for discussion and questions regarding his implementation. However, it was necessary to make major modifications and design changes for the C version, as this version needs to function with the existing CoAP code in Contiki.

The main issue when working with the SMACK protocol was that only a Java implementation existed. While this is sufficient for testing on hardware that can support the Java runtime, since the main purpose of SMACK is to solve issues with constrained devices having only a Java version is not sufficient. A version in C, or another low-level language, is necessary to evaluate the solution on actual constrained devices. Contiki was chosen as the operating systems as it is common and supported on many different types of constrained devices [7]. Additionally, Contiki was originally developed at SICS where this thesis project was performed.

Having an implementation that functions with Contiki allowed an evaluation of the SMACK solution on many different types of constrained devices. These evaluations are described in the next chapter.

4.1 Hardware

The main board that was used for evaluation and testing in this project is the Texas Instruments CC2538 board, specifically the CC2538 evaluation version. The board is shown in Figure 2-1 on page 6. Some of this board's key specifications according to its data sheet are [67]:

- ARM Cortex M3 Processor – 32MHz top clock speed
- 512 kB FLASH memory
- 32 kB RAM
- Support for several low power modes
- AES-128/256, SHA2 Hardware Encryption Engine
- 2.4 GHz IEEE 802.15.4 transceiver

The implementation was purposely kept general in order for it to function on as many of the devices that Contiki supports as possible. These devices have varying properties and varying levels of support for different low-level functions. Therefore, the SMACK implementation avoids using any board specific functionality and tries to be a general C program that can be run on as many devices as possible. This means that this implementation does not take advantage of the AES-128/256 and SHA2 Hardware Encryption Engine, this is left for future work.

The particular CC2538 board used for the experiment was set to run at 16 MHz. According to the data sheet the values in Table 4-1 can be found or calculated for the CC2538 boards [67].

Table 4-1: CC2538 test hardware key values

Name	Description	Value
Voltage	Voltage board runs on	3 volts
Radio I_RX	Current drain for radio receive	20 mA
Radio I_TX	Current drain for radio transmit	24 mA
P_TX	Power use for radio transmit	72 mW
P_RX	Power use for radio receive	60 mW
CPU	Current drain for CPU	7 mA
P_CPU	Power use for CPU	21 mW
Real Time Clock (RTC)	Clock tick rate	32768 ticks/second

4.2 Software environment used for development

Texas Instruments' Code Composer Studio (CCS) version 5.5.0 was partially used as the development environment when writing C code. CCS is a full Integrated Development Environment (IDE) and includes extensive support for debugging. A large portion of the coding was also done using the Gedit text-editor under the Contiki development VMware-image. A simple text editor was sufficient since the code that actually needed to be edited was contained in a relatively few number of files. As code already existed for simple CoAP client/server applications and the actual implementation of the CoAP protocol this existing code was used for further development. In practice, the make-files and settings were already configured and ready for compilation of the code. The client required some modifications to make it deliver keys to the main CoAP-stack and the main work took place by modifying the implementation of CoAP that comes with Contiki.

CCS was used for compiling and transferring the applications to the actual hardware. CCS not only comes with a development environment but also functionality for transferring software to constrained devices such as the CC2538 board. CCS can be used together with a USB or Joint Test Action Group (JTAG) interface[*] to transfer applications to a specific board for testing. When using a JTAG interface it is possible to do high level debugging using CCS.

4.3 SMACK C implementation

The process of creating the implementation of SMACK in C followed these steps:

1. Understand the general structure and functionality of the Contiki operating system.
2. Become familiar with the CoAP implementation in Contiki (Erbium)
3. Implement client functionality, calculating and marking packets with MACs
4. Implement a server which checks each incoming packet and verifies its MAC
5. Perform functional testing to ensure that the code behaves correctly
6. Perform performance testing to evaluate the use of SMACK with regard to the goals specified in Section 0.

[*] A JTAG interface implements the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture.

The resulting implementation is currently not available for download as there still are papers and related work ongoing at SICS based on this code. However it will likely be released in the future when papers at SICS have been published.

4.4 Energest

Contiki includes a framework for measuring the time spent in different states by a particular application. This makes it possible to measure exactly how long a Contiki program spent executing instructions or how long the radio was turned on for listening. If some key values, such as operational voltage and current drain, are known they can be used together with the timing information to calculate the energy use of a specific operation. The benefit of using this tool is that measuring and calculating power consumption (by combing the measurements with the known values) is simplified.

Using Energest is simple and requires adding only a few lines of code to an application. The results are given in units of clock ticks, but dividing these results by the number of ticks per second for a particular hardware device gives results in seconds. Table 4-2 shows the metrics that can be measured using Energest. The reliability of Energest has been evaluated in a report by Adam Dunkels [68] and his conclusions show that the testing framework adds 0.7% overhead in terms of computation time. Furthermore, the report covers practical testing comparing Energest to actual power readings from a board and shows that the estimated energy use follows the graph of the measured energy use to a high extent when looking at the specific points where samples are taken.

Table 4-2: Energest metrics

Name	**Description** (all values are measured in clock ticks)
ENERGEST_TYPE_CPU	CPU time
ENERGEST_TYPE_LPM	Time in Low Power Mode (LPM)
ENERGEST_TYPE_TRANSMIT	Radio transmission time
ENERGEST_TYPE_LISTEN	Radio reception time

5 Analysis

The analysis consists of four parts: functional evaluation, comparison of overhead between vanilla CoAP and CoAP with the SMACK extension, performance evaluation on CC2538 boards, and simulated testing on other constrained devices. The performance evaluations use actual hardware with data collected using Energest. The normal vanilla version of CoAP is compared to CoAP with the SMACK extension. The performance of each implementation is assessed in terms of latency and time taken processing messages. When it comes to code size SMACK added 55 kB to existing Contiki CoAP code (108 kB). 23.6 kB was for SHA functionality, 9.3 kB for HMAC and finally 22.1 kB for the SMACK core code.

5.1 Functional Testing

Functional testing was initially conducted using the Cooja network simulator that is included with the Instant Contiki development image. The main point of this testing was to first ensure that the implementation fulfills the basic functionality of the SMACK extension. This was accomplished in two ways. First the current Java SMACK client developed at SICS was used to interface with the simulated nodes running in Cooja and it was confirmed that the Java client could communicate correctly with the Cooja nodes. This means that the SMACK implementation on top of CoAP in Contiki was *compatible* with the external Java client and its implementation of SMACK. In addition, the traffic was analyzed using the Wireshark packet capture tool to confirm that the structure of the packets exchanged followed the SMACK protocol specification. This way of testing allowed testing of the SMACK server in C *independent* of the corresponding C client as a third party reference implementation could be used as a client to ensure that the C SMACK server meets the specifications before testing it with the developed SMACK client.

The second test performed was to simulate both a server and a client node running in Cooja and check that they could communicate with each other using the SMACK extended CoAP. Because the previous test showed that the SMACK server implementation met the specifications, the next step was to test the SMACK client implemented during this thesis project. Consequently, the client was tested against a SMACK server, both running on simulated nodes in Cooja. Additionally, the traffic was monitored by looking at debugging output from the two nodes. Both nodes were stress tested by transmitting a large number of CoAP requests (up to 10 000) to ensure that the session handling and related code did not have any issues that would appear after prolonged communication. Testing MID rollover after 65 535 messages was not tested as issues with KDC and key renewal have to be researched further. Some issues were discovered where the client or server crashed after many packets had been exchanged. These issues were corrected by making modifications to the code, among other changes the size used for some session handling data structures were reduced to ensure that the memory would not overflow when there were multiple sessions. For the CC2538 boards a maximum of 4 sessions could be supported. However old sessions are automatically cleared when all MIDs for that session have been received or if needed to make room for a new session. New incoming sessions are prioritized over older potentially inactive ones.

In addition, some testing was done to ensure that SMACK actually provides authentication of messages as it should. For instance, packets with incorrect MACs were sent and these packets were **not** accepted by a server implementing the SMACK extension. If the MAC calculated by the server from the contents of the packet does not match the MAC embedded by the client in the packet, then the message was correctly discarded. This was easily confirmed both by debug output from the server and monitoring the network using Wireshark. Basic replay protection was also tested and replayed packets were simply ignored as they are already marked as received in the bit map keeping track of packets within a session.

5.2 Comparison of packet overhead

In the case of SMACK, the number of packets required to initialize a communication session remains the same as for CoAP, except for communication with the KDC. Because SMACK has to receive keys there is periodically extra traffic generated for this. This implementation uses a KDC simulated in software, but a real setup would need some extra packets to exchange data with the KDC. One packet in each direction is sufficient for the KDC and a device to exchange the necessary information before a session starts. Ideally, the KDC should be close to the device or keys can be pre-shared between devices. It is advantageous to have the KDC close to a device to reduce latency in communicating keys. There is no handshake or other setup data exchanged between the two devices communicating using the SMACK extension, the MAC is simply added to each message by the transmitter and checked at the receiver.

As far as communication overhead is concerned SMACK does **not** increase the size of the packets exchanged. This can be seen by comparing a CoAP packet protected with SMACK and a vanilla CoAP packet, as both include the Token field (with a variable size of 0-8 bytes). Many implementations of CoAP use a 4 byte token. SMACK also uses 4 bytes for the Token field; however, it splits it into two subfields one of which contains a MAC and the other retains the same purpose as the original Token field. SMACK uses an existing field of the CoAP header to include the MAC it calculates for each packet. The great advantage of this is that the protocol does not have to be modified nor does this add any extra overhead compared to vanilla CoAP traffic. In addition to that a request protected with SMACK is backwards compatible with vanilla CoAP since the contents of the Token field will be parsed as if it contained a normal 4 byte token and the subfield containing the MAC is not parsed as such by a vanilla CoAP server. If SMACK is enabled on receiver and sender both devices have to agree on the same length for the Token field. In the case where only the client has SMACK enabled the length is not relevant since the server will simply mirror the token sent by the client. Finally if only the server is using SMACK it will reject messages from vanilla clients since they will not be including correct MAC values in the 2 reserved bytes of the Token field.

5.3 Performance Testing

Performance testing was initially performed using the hardware described in Section 4.1, i.e. a TI CC2538 board. Scenarios involving SMACK and vanilla CoAP were both evaluated. Using Energest the time in different states was measured (see Table 5-1 and Table 5-2) and the power consumption was calculated using power information from the data sheets of the board.

The time it takes to perform the MAC calculation was measured both for steady-state operation and also for the first SMACK packet (that establishes a session and generates keys). The whole time taken from the stack receiving a CoAP packet to when it is delivered to the receiving application was measured. This was also done for both steady-state and initial transactions. As vanilla CoAP does not use a MAC the portion of the code calculating it was not tested for vanilla CoAP. Instead only steady-state and initial transactions were compared. In practice, the different tests were accomplished by controlling where in the source code of the Contiki CoAP stack the Energest start and end measurements calls were placed. As mentioned Energest needs only a few lines at the start and stop of the blocks of code of interest to measure the number of clock ticks during which the different components were active.

Table 5-1: SMACK measurements

Start	Stop	State	Name
CoAP request reception	CoAP request delivery to application	1st transaction	A
Right before MAC check	Right after MAC check	1st transaction	B
CoAP request reception	CoAP request delivery to application	Steady-state	C
Right before MAC check	Right after MAC check	Steady-state	D

Table 5-2: Vanilla CoAP measurements

Start	Stop	State	Name
CoAP request reception	CoAP request delivery to application	1st transaction	E
CoAP request reception	CoAP request delivery to application	Steady-state	F

The duration between initial request and reply was measured when using unmodified CoAP and compared to that of CoAP with SMACK. The detailed results from this testing can be seen in Appendix A. All values from Energest of LPM and TRANSMIT were **zero**. This is because low-power mode was not enabled as the processor was performing calculations and it was not in a resting state. In addition to that the radio was listening for incoming traffic and not in transmit mode. The actual time in seconds calculated from the ticks can be found by dividing the ticks by 32 768 since that is the frequency of the internal clock. To get the actual energy usage the values in Table 4-1 can be used. The formula used is the following:

$$E = P_{CPU} \times \frac{CPU\ ticks}{RTC\ frequency} + P_{RX} \times \frac{LISTEN\ ticks}{RTC\ frequency}$$

The data in Table 5-3 show an overview of the test results with a confidence interval of 95% applied to the values calculated.

Table 5-3: Energy statistics

Test	Energy (µJ) 95% confidence interval	Client time (ms) 95% confidence interval
SMACK full request 1st transaction (A)	316.22 ±0.97	43.32 ±2.02
SMACK MAC check 1st transaction (B)	313.00 ±0.87	44.75 ±1.74
SMACK full request steady-state (C)	49.04 ±0.74	29.47 ±0.44
SMACK MAC check steady-state (D)	45.73 ±0.83	29.02 ±0.30
Vanilla CoAP full request 1st transaction (E)	5.44 ±0.41	40.31 ±1.47
Vanilla CoAP full request steady-state (F)	5.72 ±0.46	28.39 ±0.32

5.4 Testing on other constrained devices

Hardware wise the code was only tested on the CC2538 devices and confirmed to function on those boards. However, using Cooja the code was also tested in a simulated environment for the Z1 [69] and WiSMote [70] type boards. On those boards the code functions without problems and can be comprehensively tested from a networking and software point of view to the extent that is possible in a simulation. Because Cooja supports simulated network traffic and even makes it possible to connect to boards inside the simulation from the host computer the code could also be tested with the Java version of SMACK and the Californium CoAP implementation. This means that the code could be tested with the same tools, the same network requests, and same client-side code used for the testing on the CC2538 hardware.

5.5 Chapter summary

From the packet overhead point of view SMACK does not add any additional packet overhead beyond what vanilla CoAP utilizes. However, SMACK adds some extra packets for initializing

communication due to the communications with a KDC that distributes keying material. Initializing SMACK communication uses significantly more energy compared to a vanilla CoAP request. Even in steady-state communication SMACK increases the energy usage of the constrained device. A client using SMACK to communicate with a server does not experience any significant slowdown; as requests using SMACK compared to vanilla CoAP experience at most only a few milliseconds of extra latency. Security wise SMACK provides authentication of messages and ensures that packets with an incorrect MAC are not accepted. An incorrect MAC can be both due to a packet being modified in transit or having the wrong keys used for the MAC calculation.

6 Conclusions and Future work

This chapter contains the conclusions drawn from the work performed during this thesis project. It also covers interesting aspects that can be explored in the future and suggests some of the best directions to continue work on this problem. There are some aspects that were outside the scope of this report and also some aspects that were not investigated further due to the bounded duration of this thesis project. Finally, this chapter also includes some reflections regarding ethical, environmental, and social aspects of this work. These issues should also be taken into account when considering how to proceed with this topic. These considerations are important to ensure that the work done has value and is a good place to invest research resources as compared to other potential solutions and areas.

6.1 Conclusions

When comparing SMACK to the technologies described in Chapter 2 and elsewhere one clear benefit of SMACK compared to other solutions is its low overhead. For comparison, the "CoAP security options" proposed by Yegin adds up to 30 bytes of overhead per packet which may be unacceptable in constrained networks. By reusing parts of the CoAP header, a MAC could be added without expanding the packet's size. SMACK requires limited memory for replay protection, only using one bit per packet in a session. As a result, a session size of 127 requires 16 bytes for replay protection. Another benefit of SMACK is that it provides end-to-end security in contrast to layer 2 security solutions, such as IEEE 802.15.4. Although there are also drawbacks to this method, as it is not able to protect lower layer headers.

SMACK requires more energy compared to vanilla CoAP. This is not entirely surprising as additional calculations, in the form of the MAC calculation, are added as compared to vanilla CoAP. An interesting factor is how SMACK compares in resource use to other alternatives. As mentioned in Section 2.10 a paper found DTLS to be unsuitable for constrained devices due to its high resource use. However, in reality DTLS is currently used for securing communication on constrained devices and it is the recommended option for adding security to CoAP. Many other solutions such as SRTP are more costly when it comes to calculations required. For instance SRTP requires one HMAC calculation for each packet while SMACK only requires one every 16^{th} packet (per default). SMACK performs a full HMAC calculation for every 16^{th} packet and instead does the more lightweight Galois calculation of the MAC for each packet. This means that the processing induced by HMAC calculations will be significantly less in the case of SMACK, thereby saving some computational resources which can translate into saving power.

Another result of using SMACK is that it enables a device to identify unsolicited traffic that is not properly authenticated. This gives the option of rejecting this traffic and possibly saving resources. In more advanced attacks where computationally expensive operations can be triggered on a host the protection SMACK affords can be useful even though it adds some computations as compared to vanilla CoAP. Further development of SMACK and possibly utilizing hardware encryption engines can reduce the power consumed in this authentication. In addition, using SMACK and having the ability to identify unauthenticated messages allows deploying proactive strategies. For instance, the attacker can be blocked at an earlier hop or the listening device can instruct the radio to use a different frequency or stop listening entirely. Compared to vanilla CoAP using SMACK means a node can distinguish between incoming legitimate messages and spurious ones.

The main goal of implementing a version of the SMACK extension using C for the Contiki platform was accomplished. In addition, the implementation was successfully tested both on the Cooja simulator for various device as well as on CC2538 boards. The implementation was experimentally evaluated on these boards and compared to the vanilla version of CoAP. Even though a Java version of SMACK existed, a new implementation had to be written from scratch to fit into the architecture used to implement CoAP on Contiki. C is sufficiently different from Java that a complete rewrite was necessary. Different existing options for authentication and security on the IoT were described and their various benefits and drawbacks were described. Some further testing in practical experiments on

other hardware, testing with an attacker, and experimentally evaluating alternative security solutions are interesting avenues but were considered future work.

Developing software on Contiki and adapting to the difference when writing code for constrained devices can take some time. For instance, testing the code on actual hardware can be a laborious procedure since the code has to be compiled and transferred to the memory of the boards. For the CC2538 platform the procedure of compiling and transferring an application to the hardware (the test boards) can take 5 minutes. Unfortunately, many of the development tools for theses device are lacking in functionality and polish. As a result, the best approach is to test and develop code using a simulator, such as Cooja. However, in some cases the specific board type being used does not exist in Cooja, so testing will have to use a different simulated board which can cause additional problems. However, it is important to periodically test the application on actual hardware to ensure that it actually functions as it should. The Cooja simulator is good, but unfortunately some inconsistencies can appear between real world performance and the simulation. Debugging support in Cooja is lacking and problems in the code can cause it to crash.

6.2 Future work

One interesting aspect for future work is to attempt to mitigate cases when the radio is simply overwhelmed with traffic. In these cases, it might be appropriate to simply power down the radio and ignore all traffic for a fixed period of time. If a device believes that it is under attack, then powering down the radio avoids using any power for receiving radio signals and parsing of messages. Using the SMACK extension allows the node to detect some forms of DoS attacks and initiate countermeasures. For example, a device could have a rule that when 100 packets with an invalid MAC have been received the radio should be powered down for 1 hour. Of course, this leads to a very simple denial of service attack, where the attacker simply sends many invalid packets to cause the device to power down its radio for an hour – thus preventing the device from carrying out its actual purpose for legitimate users. Alternatively, a device could stop listening to a specific frequency, network interface, or transmitter. However, in practice the number of frequencies that the device can operate in is limited and the attacker can utilize another address to continue the attack.

Of course, the rule for when to power down or stop listening to a malicious transmitter needs to be carefully thought out and a study would need to be done to find an appropriate rule. Since powering down the radio means that no messages will be received and the device will be non-functional for a period of time, this has to be weighed against the potential benefits of doing so. For an individual sensor node powering down the radio conserves battery power (and thereby enables the continued operation of the node at some future time) which may be better than succumbing to an attack and permanently stop functioning. Future research should address the many complex tradeoffs that exist. How to prioritize degrees of functionality versus operating lifetime remains an open question.

Another area to investigate is to evaluate different scenarios with an attacker in the network. Ensuring that legitimate clients can access a service while it is under attack is important. Research should examine how this can be ensured and to what extent SMACK provides this capability. The tests done as part of this thesis project only show SMACK functioning with a basic client server setup *without* an adversary. Adversaries can employ different tactics to discover which methods of attack are the most effective at bypassing SMACK's protection. Ideally, SMACK could be adapted to provide better protection and countermeasures against the most successful attack strategies. Evaluating SMACK against real attacks and strategies is important for SMACK to become a more robust and reliable mechanism. Advanced attacks employing modified replayed packets or other techniques should be tested and evaluated against SMACK.

It would also be interesting to run SMACK on additional or more highly constrained devices than those described in Section 4.1. One benefit of SMACK is that it is a relatively simple system hence it does not require much processing power or memory to operate. This is an advantage compared to other solutions and it would be interesting to investigate if and when SMACK would have a role, i.e., in which settings other solutions simply cannot be implemented due to the resource constraints of the platform. Implementation details such as how to best use SMACK together with a KDC and how to

best deal with changing keys after 65 536 messages (when the MID loops back to 0) should also be considered deeper.

A future effort should take advantage of any AES-128/256 or SHA2 Hardware Encryption Engine that the device supports as part of the SMACK implementation. As per Section 2.6.2, it would be interesting to exploit IEEE 802.15.4 encryption and authentication in order to reduce the power consumption at the non-gateway wireless nodes. As noted previously this would break the end-to-end security, but might reduce both bandwidth requirements and power consumption of the constrained nodes. Also putting the authentication; related calculations and power drain on a gateway node can save the internal nodes the burden of performing this functionality. This can be especially beneficial when the gateway node is more powerful than the rest of the nodes in the network or if it already operates as a sink node in a sensor network.

6.3 Required reflections

The growth of sensor based systems and constrained devices has been large during the last several years and this trend is predicted to continue in the coming years [3]. Because of this, the attack surface of sensor based systems and constrained devices will also grow and hackers and malicious individuals will target these devices. Compounding this problem is that security solutions for constrained devices are not as mature as they are for conventional devices and the Internet as a whole. Attacks against sensor systems and other similar device are a potential threat that needs to be taken seriously. Manipulation or damaging such systems can have serious consequences, especially for industrial, medical, and safety applications. Both monitoring and control systems are increasingly automated and in many cases these applications are deployed using constrained devices.

From an environmental point of view, reducing power consumption is beneficial. Many types of batteries contain harmful chemicals and damage the environment, hence many governmental organizations including the European Union (EU) have created regulations concerning them [71]. If the number of batteries used can be reduced this is beneficial to the environment. Preventing battery exhaustion attacks caused intentionally by attackers or misconfigured devices can potentially reduce the number of batteries that will be used. Sensor nodes are frequently used to monitor the environment, for example measuring pollution near a road. Improving the performance, reliability, and protecting these nodes against remote tampering with the sensor platform can help provide better and longer term environmental studies.

Reducing waste is also beneficial from an economical point of view. Preventing attacks and damage to sensor nodes and other constrained devices is highly desirable since repairs and maintenance to such a device is generally costly. Furthermore, incorrect sensor data can damage equipment or produce incorrect results leading either to a need for replacing equipment or in some cases a need to redo experiments. There are also possible liability issues where companies that do not provide the maximum security possible to their customers can be held responsible for any damage caused by negligent security systems. The public relations impact on a company from having one of their systems attacked can cause financial repercussions and loss of trust.

From an ethical point of view, adding protection and authentication to network traffic is beneficial for all involved parties. This is especially true today as these issues are frequently discussed in the news; hence protecting systems is often a high priority for companies and knowledgeable individuals. If a simple modification can be done to reduce the impact of attacks or prevent certain types of attacks, then there are many who feel that this modification should be adopted. Today it is increasingly common to see individuals attacking systems for either political or ideological reasons. Some of these attackers may feel they are acting completely ethically. Today there are even states who attack the infrastructures of others via hacking and other means. This has recently fostered discussion on the ethics of cyberwarfare.

An important goal of SMACK is to provide partial protection for sensor systems from many of these issues. In particular, battery exhaustion attacks by premeditated attackers, ill-configured networks, or excessive traffic from some nodes should be possible to mitigate by using the SMACK extension of CoAP. By implementing SMACK the systems should become more robust and less

vulnerable to certain types of attacks. The current incarnation of SMACK does not meet all these goals but further development can improve the protocol and aim to alleviate some or all of these problems. SMACK is also backwards compatible with existing implementations of CoAP, as it works within the limits of the protocol specification. This thesis can contribute to the research area of security in the IoT which is a contemporary and continuously evolving one. In fact, CoAP only recently went through the final steps of standardization and was released as RFC7252 during June 2014 [5].

References

[1] ‘About SICS Swedish ICT’, *About SICS | SICS*. [Online]. Available: https://www.sics.se/about-sics. [Accessed: 02-Mar-2014]

[2] K. Ashton, ‘That “Internet of Things” Thing’, *RFID Journal*, New York, NY, 22-Jun-2009 [Online]. Available: http://www.rfidjournal.com/articles/view?4986. [Accessed: 02-Mar-2014]

[3] ABI Research, ‘More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020’, *ABI Research press release*, London, United Kingdom, 09-May-2013 [Online]. Available: https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne. [Accessed: 02-Mar-2014]

[4] Qijun Gu and Peng Liu, ‘Denial of Service Attacks’, *Handbook of Computer Networks: Distributed Networks, Network Planning, Control, Management, and New Trends and Applications*, vol. 3, Jun. 2007 [Online]. DOI: 10.1002/9781118256107.ch29

[5] Z. Shelby, K. Hartke, and C. Bormann, ‘The Constrained Application Protocol (CoAP)’, *Internet Request for Comments*, vol. RFC 7252 (Proposed Standard), Jun. 2014 [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[6] IEEE Computer Society, ‘Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)’, Institute of Electrical and Electronics Engineers, New York, NY, IEEE Standard 0-7381-4997-7, Sep. 2006 [Online]. Available: https://standards.ieee.org/getieee802/download/802.15.4-2011.pdf

[7] ‘Contiki Hardware’, *Contiki Hardware*. [Online]. Available: http://www.contiki-os.org/hardware.html. [Accessed: 02-Mar-2014]

[8] M. Stemm and R. H. Katz, ‘Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices’, *IEICE Transactions on Communication*, vol. E80-B, no. 8, pp. 1125–1131, Aug. 1997 [Online]. Available: http://www.cs.colorado.edu/~rhan/CSCI_7143_002_Fall_2001/Papers/Stemm97_EnergyConsumptionNetworkInterface.pdf

[9] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, ‘TAG: a Tiny AGregation Service for Ad-Hoc Sensor Networks’, in *5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, Massachusetts, USA, 2012 [Online]. Available: http://db.lcs.mit.edu/madden/html/madden_tag.pdf

[10] ‘Texas Instruments - CC2538dk - CC2538, Zigbee/802.15.4, Dev Kit’, *CC2538dk - Texas Instruments - CC2538, Zigbee/802.15.4, Dev Kit | Farnell Sverige*. [Online]. Available: http://se.farnell.com/texas-instruments/cc2538dk/cc2538-zigbee-802-15-4-dev-kit/dp/2356505. [Accessed: 02-Mar-2014]

[11] ‘Tmote Sky - Ultra low power IEEE 802.15.4 compliant wireless sensor module’. Moteiv Corporation, 02-Jun-2006 [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

[12] Z. Shelby, Sensinode, K. Hartke, and C. Bormann, ‘Constrained Application Protocol (CoAP)’, *Internet Draft*, vol. draft-ietf-core-coap-18, p. 25, Jun. 2013 [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-coap-18. [Accessed: 02-Mar-2014]

[13] Matthias Kovatsch, ‘Matthias Kovatsch’, *Matthias Kovatsch*, 08-May-2014. [Online]. Available: http://people.inf.ethz.ch/mkovatsc/. [Accessed: 02-Mar-2014]

[14] M. Kovatsch, ‘GitHub Repositories’. 2014 [Online]. Available: http://github.com/mkovatsc?tab=repositories

[15] R. Fielding, ‘Architectural Styles and the Design of Network-based Software Architectures’, Doctoral dissertation, University of California, Irvine, 2000 [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. [Accessed: 02-Mar-2014]

[16] N. Mitra and Y. Lafon, ‘SOAP Version 1.2 Part 0: Primer (Second Edition)’, *W3C Recommendation*, Apr. 2007 [Online]. Available: http://www.w3.org/TR/soap12-part0/#L1153

[17] A. Yegin and Z. Shelby, ‘CoAP Security Options’, *Internet Draft*, vol. draft-yegin-coap-security-options-00, Oct. 2011 [Online]. Available: https://tools.ietf.org/html/draft-yegin-coap-security-options-00

[18] D. Whiting, R. Housley, and N. Ferguson, ‘Counter with CBC-MAC (CCM)’, *Internet Request for Comments*, vol. RFC 3610 (Informational), Sep. 2003 [Online]. Available: http://www.rfc-editor.org/rfc/rfc3610.txt
[19] M. Dworkin, ‘Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality’, National Institute of Standards and Technology, Gaithersburg, MD, NIST Special Publication 800-38C, May 2004 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf
[20] J. Granjal, E. Monteiro, and J. S. Silva, ‘Application-Layer Security for the WoT: Extending CoAP to Support End-to-End Message Security for Internet-Integrated Sensing Applications’, in *Wired/Wireless Internet Communication*, vol. 7889, V. Tsaoussidis, A. J. Kassler, Y. Koucheryavy, and A. Mellouk, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 140–153 [Online]. Available: http://link.springer.com/10.1007/978-3-642-38401-1_11. [Accessed: 04-Mar-2014]
[21] M. Kovatsch, ‘Californium (Cf) CoAP framework in Java’, *Californium (Cf) CoAP framework - Java CoAP Implementation*, 05-Feb-2014. [Online]. Available: http://people.inf.ethz.ch/mkovatsc/californium.php. [Accessed: 02-Mar-2014]
[22] ‘ETSI CTI Plugtests Guide First Draft V0.0.15’, Sophia Antipolis, France, Mar. 2012 [Online]. Available: http://www.etsi.org/plugtests/CoAP/Document/CoAP_TestDescriptions_v015.pdf. [Accessed: 02-Mar-2014]
[23] D. Pauli and D. Im Obersteg, ‘Californium’, Lab Project, Swiss Federal Institute of Technology Zurich, Zurich, 2011 [Online]. Available: http://people.inf.ethz.ch/mkovatsc/resources/californium/cf-thesis.pdf
[24] Dennis Morse, Roland McGrath, and Mike Frysinger, ‘make - GNU make utility to maintain groups of programs’, *UNIX man pages : make ()*, 22-Aug-1989. [Online]. Available: http://unixhelp.ed.ac.uk/CGI/man-cgi?make. [Accessed: 02-Mar-2014]
[25] Apache Maven Project, ‘Introduction to the POM’, *Maven - Introduction to the POM*, 21-May-2014. [Online]. Available: http://maven.apache.org/guides/introduction/introduction-to-the-pom.html. [Accessed: 02-Mar-2014]
[26] Mathilde Durvy, Julien Abeillé, Patrick Wetterwald, Colin O’Flynn, Blake Leverett, Eric Gnoske, Michael Vidales, Geoff Mulligan, Nicolas Tsiftes, Niclas Finne, and Adam Dunkels, ‘Making Sensor Networks IPv6 Ready’, in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, Raleigh, NC, USA, 2008, pp. 421–422 [Online]. DOI: 10.1145/1460412.1460483
[27] A. Dunkels, O. Schmidt, T. Voigt, and A. Muneeb, ‘Protothreads: Simplifying event-driven programming of memory constrained embedded systems’, in *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems*, Boulder, Colorado, USA, 2006 [Online]. Available: http://dunkels.com/adam/dunkels06protothreads.ppt. [Accessed: 02-Mar-2014]
[28] A. Dunkels, ‘Contiki: Bringing IP to Sensor Networks’, *ERCIM News*, no. 76, pp. 59–60, Jan-2009 [Online]. Available: http://ercim-news.ercim.eu/images/stories/EN76/EN76-web.pdf
[29] M. Kovatsch, S. Duquennoy, and A. Dunkels, ‘A Low-Power CoAP for Contiki’, in *Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems (MASS ’11)*, 2011, pp. 855 – 860 [Online]. DOI: 10.1109/MASS.2011.100
[30] ‘Get Started with Contiki’, *Get Started with Contiki, Instant Contiki and Cooja*. [Online]. Available: http://www.contiki-os.org/start#simulation. [Accessed: 02-Mar-2014]
[31] Z. Shelby, *6LoWPAN: the wireless embedded internet*. Chichester, U.K: J. Wiley, 2009.
[32] J. Hui and P. Thubert, ‘Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks’, *Internet Request for Comments*, vol. RFC 6282 (Proposed Standard), pp. 10–11, Sep. 2011 [Online]. Available: http://www.rfc-editor.org/rfc/rfc6282.txt
[33] S. Raza, D. Trabalza, and T. Voigt, ‘6LoWPAN Compressed DTLS for CoAP’, presented at the IEEE 8th International Conference on Distributed Computing in Sensor Systems (DCOSS), 2012, 2012, pp. 287–289 [Online]. DOI: 10.1109/DCOSS.2012.55
[34] N. Sastry and D. Wagner, ‘Security Considerations for IEEE 802.15.4 Networks’, in *WiSe ’04 Proceedings of the 3rd ACM workshop on Wireless security*, Philadelphia, Pennsylvania, 2004 [Online]. DOI: 10.1145/1023646.1023654

[35] Z. Shelby and C. Bormann, ‘Layer 3 mechanisms’, in *6LoWPAN: The Wireless Embedded Internet*, 1st ed., Chichester, U.K: J. Wiley, 2009, p. 87 [Online]. Available: http://elektro.upi.edu/pustaka.elektro/Wireless%20Sensor%20Network/6LoWPAN.pdf
[36] S. Kent and R. Atkinson, ‘Security Architecture for the Internet Protocol’, *Internet Request for Comments*, vol. RFC 2401 (Proposed Standard), Nov. 1998 [Online]. Available: http://www.rfc-editor.org/rfc/rfc2401.txt
[37] S. Kent and K. Seo, ‘Security Architecture for the Internet Protocol’, *Internet Request for Comments*, vol. RFC 4301 (Proposed Standard), p. 85, Dec. 2005 [Online]. Available: http://www.rfc-editor.org/rfc/rfc4301.txt
[38] John Ioannidis and Matt Blaze, ‘Architecture and Implementation of Network-layer Security Under Unix’, in *Proceedings of USENIX Security Symposium*, Santa Clara, California, USA, 1993.
[39] P. Lambert, ‘Minutes of the Internet Protocol Security Protocol Working Group (IPSEC)’, Toronto, Canada, Meeting report, Jul. 1994 [Online]. Available: ftp://ftp.ietf.org/ietf-online-proceedings/94jul/area.and.wg.reports/sec/ipsec/ipsec-minutes-94jul.txt. [Accessed: 02-Jun-2014]
[40] Sheila Frankel, Karen Kent, Ryan Lewkowski, Angela D. Orebaugh, Ronald W. Ritchey, and Steven R. Sharma, ‘Guide to IPsec VPNs - Recommendations of the National Institute of Standards and Technology’, U.S. Department of Commerce, Gaithersburg, MD, NIST Special Publication 800-77, Dec. 2005 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-77/sp800-77.pdf
[41] E. Jankiewicz, J. Loughney, and T. Narten, ‘IPv6 Node Requirements’, *Internet Request for Comments*, vol. RFC 6434 (Informational), p. 18, Dec. 2011 [Online]. Available: http://www.rfc-editor.org/rfc/rfc6434.txt
[42] Christos Xenakis, Nikolaos Laoutaris, Lazaros Marakos, and Ioannis Stavrakakis, ‘A generic characterization of the overheads imposed by IPsec and associated cryptographic algorithms’, University of Athens, Athens, 50, May 2005 [Online]. Available: http://www.cse.msstate.edu/~ramkumar/ipsec_overheads.pdf
[43] S. Park, W. Haddad, S. Chakrabarti, J. Laganier, and K. Kim, ‘IPv6 over Low Power WPAN Security Analysis’, *Internet Draft*, vol. draft-daniel-6lowpan-security-analysis-05, p. 16, Mar. 2011 [Online]. Available: http://www.potaroo.net/ietf/all-ids/draft-daniel-6lowpan-security-analysis-05.txt. [Accessed: 02-Mar-2014]
[44] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, ‘The Secure Real-time Transport Protocol (SRTP)’, *Internet Request for Comments*, vol. RFC 3711 (Proposed Standard), Mar. 2004 [Online]. Available: http://www.rfc-editor.org/rfc/rfc3711.txt
[45] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, ‘RTP: A Transport Protocol for Real-Time Applications’, *Internet Request for Comments*, vol. RFC 1889 (Proposed Standard), Jan. 1996 [Online]. Available: http://www.rfc-editor.org/rfc/rfc1889.txt
[46] H. Krawczyk, M. Bellare, and R. Canetti, ‘HMAC: Keyed-Hashing for Message Authentication’, *Internet Request for Comments*, vol. RFC 2104 (Informational), Feb. 1997 [Online]. Available: http://www.rfc-editor.org/rfc/rfc2104.txt. [Accessed: 07-May-2014]
[47] S. Garg, N. Singh, and T. Tsai, ‘Short Paper: Schemes for Enhancing the Denial-of-Service Tolerance of SRTP’, in *First International Conference on*, Athens, Greece, 2005, pp. 409 – 411 [Online]. DOI: 10.1109/SECURECOMM.2005.48
[48] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman, ‘MIKEY: Multimedia Internet KEYing’, *Internet Request for Comments*, vol. RFC 3830 (Proposed Standard), Aug. 2004 [Online]. Available: http://www.rfc-editor.org/rfc/rfc3830.txt
[49] J. Arkko, F. Lindholm, M. Naslund, K. Norrman, and E. Carrara, ‘Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)’, *Internet Request for Comments*, vol. RFC 4567 (Proposed Standard), Jul. 2006 [Online]. Available: http://www.rfc-editor.org/rfc/rfc4567.txt
[50] T. Dierks and E. Rescorla, ‘The Transport Layer Security (TLS) Protocol Version 1.2’, *Internet Request for Comments*, vol. RFC 5246 (Proposed Standard), p. 14, Aug. 2008 [Online]. Available: http://www.rfc-editor.org/rfc/rfc5246.txt. [Accessed: 02-May-2014]

[51] E. Rescorla and N. Modadugu, ‘Datagram Transport Layer Security’, *Internet Request for Comments*, vol. RFC 4347 (Proposed Standard), Apr. 2006 [Online]. Available: http://www.rfc-editor.org/rfc/rfc4347.txt
[52] E. Rescorla and N. Modadugu, ‘Datagram Transport Layer Security Version 1.2’, *Internet Request for Comments*, vol. RFC 6347 (Proposed Standard), Jan. 2012 [Online]. Available: http://www.rfc-editor.org/rfc/rfc6347.txt. [Accessed: 02-Mar-2014]
[53] F. Chunyan, ‘TCP/UDP Basics’, Canada [Online]. Available: http://users.encs.concordia.ca/~glitho/F09_TCP_UDP.pdf
[54] Bhargavan, Karthikeyan and Fournet, Cedric and Kohlweiss, Markulf and Pironti, Alfredo and Strub, and Pierre-Yves, ‘Implementing TLS with Verified Cryptographic Security’, presented at the 2013 IEEE Symposium on Security and Privacy (SP), Berkeley, CA, USA, 2013, pp. 445–459 [Online]. DOI: 10.1109/SP.2013.37
[55] Stefan Jucker, ‘Securing the Constrained Application Protocol’, Master’s Thesis, ETH Zurich, Zurich, 2012 [Online]. Available: http://people.inf.ethz.ch/mkovatsc/resources/californium/cf-dtls-thesis.pdf
[56] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, ‘Lithe: Lightweight Secure CoAP for the Internet of Things’, *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3711–3720, Oct. 2013 [Online]. DOI: 10.1109/JSEN.2013.2277656
[57] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar, and K. Wehrle, ‘Security Challenges in the IP-based Internet of Things’, *Wireless Personal Communications*, vol. 61, no. 3, pp. 527–542, Sep. 2011 [Online]. DOI: 10.1007/s11277-011-0385-5
[58] T. Marco, ‘Talk by Marco Tiloca’, Amsterdam, CWI, Room L017, 13-Dec-2013 [Online]. Available: http://projects.cwi.nl/crypto/risc.html. [Accessed: 02-Mar-2014]
[59] Çetin Kaya Koç, ‘Message Authentication’, University of California Santa Barbara [Online]. Available: http://cs.ucsb.edu/~koc/ccs130h/notes/mac2.pdf
[60] M. Bellare, R. Canetti, and H. Krawczyk, ‘Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security’, p. 3, Oct. 2005 [Online]. Available: http://cseweb.ucsd.edu/~mihir/papers/cascade.pdf
[61] Xiaoyun Wang and Hongbo Yu, ‘How to Break MD5 and Other Hash Functions’, in *EUROCRYPT 2005*, Aarhus, Denmark, 2005 [Online]. DOI: 10.1007/11426639_2
[62] M. Bellare, R. Canetti, and H. Krawczyk, ‘Keying Hash Functions for Message Authentication’, in *Crypto 96 Proceedings*, Santa Barbara, California, USA, 1996 [Online]. Available: http://cseweb.ucsd.edu/~mihir/papers/kmd5.pdf
[63] Jorge Castiñeira Moreira and Patrick Guy Farrell, ‘Appendix B: Galois Fields GF(q)’, in *Essentials of Error-Control Coding*, 1st ed., Wiley, 2006 [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/9780470035726.app2/pdf
[64] J. Torres-Jimenez, N. Rangel-Valdez, A. L. Gonzalez-Hernandez, and H. Avila-George, ‘Construction of logarithm tables for Galois Fields’, *International Journal of Mathematical Education in Science and Technology*, vol. 42, no. 1, pp. 91–102, Feb. 2010 [Online]. DOI: 10.1080/0020739X.2010.510215
[65] James S. Plank, ‘Fast Galois Field Arithmetic Library in C/C++’. [Online]. Available: http://web.eecs.utk.edu/~plank/plank/papers/CS-07-593/
[66] S. Malladi, J. Alves-Foss, and R. B. Heckendorn, ‘On Preventing Replay Attacks on Security Protocols’, University of Idaho, Moscow, ID, USA [Online]. Available: http://www.researchgate.net/publication/2837470_On_Preventing_Replay_Attacks_on_Security_Protocols
[67] ‘CC2538 - A Powerful System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN and ZigBee Applications’. Texas Instruments, Dec-2012 [Online]. Available: http://www.ti.com/lit/gpn/cc2538
[68] Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He, ‘Software-based On-line Energy Estimation for Sensor Nodes’, Swedish Institute of Computer Science [Online]. Available: http://dunkels.com/adam/dunkels07softwarebased.pdf
[69] Zolertia, ‘Z1 Low-Power WSN Platform’. Zolertia [Online]. Available: http://zolertia.com/sites/default/files/Zolertia-Z1-Brochure.pdf

[70] Arago Systems, ‘WiSMote - IPv6 platform for Wireless Sensor Networks R&D’. Arago Systems [Online]. Available: http://www.aragosystems.com/images/stories/WiSMote/Doc/wismote_en.pdf

[71] European Parliament and Council of the European Union, ‘Directive 2006/66/EC of the European Parliament and of the Council on batteries and accumulators and waste batteries and accumulators and repealing Directive 91/157/EEC’, *OJ*, vol. 49, no. L266, Sep. 2006 [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32006L0066&from=EN

Appendix A. Detailed results

Appendix table A-1: SMACK full request 1st transaction (A)

Test	CPU (ticks)	LISTEN (ticks)	Client time (ms)	Energy (µJ)
1	129	129	45.372320	318.8782
2	127	127	46.492054	313.9343
3	128	129	48.488310	318.2373
4	128	129	49.536816	318.2373
5	128	127	49.640880	314.5752
6	128	128	39.126153	316.4063
7	128	127	40.549250	314.5752
8	128	129	39.731519	318.2373
9	127	127	38.662179	313.9343
10	129	128	39.701068	317.0471
11	128	129	47.206422	318.2373
12	128	128	42.439143	316.4063
13	128	128	46.964981	316.4063
14	127	126	38.568382	312.1033
15	128	128	39.001957	316.4063
16	128	128	46.730383	316.4063
17	127	126	39.952008	312.1033
18	129	128	49.396929	317.0471
19	129	129	39.375747	318.8782
20	128	128	39.457462	316.4063

Appendix table A-2: SMACK MAC check 1st transaction (B)

Test	CPU (ticks)	LISTEN (ticks)	Client time (ms)	Energy (µJ)
1	128	127	39.695271	314.5752
2	126	126	49.269812	311.4624
3	125	125	39.142058	308.9905
4	127	128	45.892009	315.7654
5	126	126	38.773785	311.4624
6	127	127	48.853489	313.9343
7	127	127	39.299551	313.9343
8	126	126	41.948121	311.4624
9	127	127	46.099647	313.9343
10	127	127	46.726752	313.9343
11	127	126	46.520161	312.1033
12	127	128	47.393846	315.7654
13	127	127	39.419678	313.9343
14	126	126	48.452041	311.4624
15	127	126	42.934851	312.1033
16	128	127	45.814485	314.5752
17	127	127	46.373495	313.9343
18	127	127	46.284932	313.9343
19	126	127	49.778397	313.2935
20	126	125	46.282492	309.6313

Appendix table A-3: SMACK full request steady-state (C)

Test	CPU (ticks)	LISTEN (ticks)	Client time (ms)	Energy (µJ)
1	19	19	28.461930	46.96655
2	20	20	29.093210	49.43848
3	22	21	28.795830	52.55127
4	19	19	30.494241	46.96655
5	20	20	29.145955	49.43848
6	19	19	31.139220	46.96655
7	20	20	31.239230	49.43848
8	20	20	28.949352	49.43848
9	20	20	29.215587	49.43848
10	21	20	30.112565	50.07935
11	19	19	31.168349	46.96655
12	20	20	30.184146	49.43848
13	19	19	28.414438	46.96655
14	20	20	28.792418	49.43848
15	21	20	29.173403	50.07935
16	20	20	29.888857	49.43848
17	20	21	28.479949	51.26953
18	20	20	29.251555	49.43848
19	19	19	28.494056	46.96655
20	21	20	28.895504	50.07935

Appendix table A-4: SMACK MAC check steady-state (D)

Test	CPU (ticks)	LISTEN (ticks)	Client time (ms)	Energy (µJ)
1	19	18	28.257435	45.1355
2	18	18	28.502297	44.49463
3	19	19	28.854716	46.96655
4	18	18	28.591555	44.49463
5	19	19	28.923440	46.96655
6	18	18	28.547190	44.49463
7	19	19	29.066824	46.96655
8	18	18	28.937430	44.49463
9	19	19	28.423447	46.96655
10	20	20	30.988996	49.43848
11	18	19	29.780253	46.32568
12	18	17	29.230812	42.66357
13	19	19	29.207136	46.96655
14	19	19	29.084704	46.96655
15	19	19	28.813161	46.96655
16	18	18	28.255129	44.49463
17	17	17	29.122000	42.02271
18	18	19	28.912475	46.32568
19	19	19	29.922730	46.96655
20	18	18	28.912965	44.49463

Appendix table A-5: Vanilla CoAP full request 1st transaction (E)

Test	CPU (ticks)	LISTEN (ticks)	Client time (ms)	Energy (µJ)
1	2	2	35.326839	4.943848
2	2	2	42.886738	4.943848
3	3	3	42.447995	7.415771
4	2	2	42.448066	4.943848
5	2	2	42.941914	4.943848
6	2	2	42.008624	4.943848
7	2	3	34.963874	6.774902
8	3	2	41.038543	5.584717
9	2	2	42.107729	4.943848
10	2	2	40.472046	4.943848
11	2	2	42.123713	4.943848
12	2	2	42.134863	4.943848
13	2	3	42.430535	6.774902
14	2	2	34.890959	4.943848
15	3	3	41.660256	7.415771
16	2	2	40.886314	4.943848
17	2	2	35.690293	4.943848
18	3	2	34.809036	5.584717
19	2	2	42.265361	4.943848
20	2	2	42.632726	4.943848

Appendix table A-6: Vanilla CoAP full request steady-state (F)

Test	CPU (ticks)	LISTEN (ticks)	Client time (ms)	Energy (µJ)
1	2	3	27.997135	6.774902
2	3	2	28.012500	5.584717
3	3	3	28.524019	7.415771
4	3	3	28.951866	7.415771
5	2	2	29.291832	4.943848
6	2	2	28.062786	4.943848
7	2	3	27.743472	6.774902
8	3	3	28.428942	7.415771
9	2	2	27.472838	4.943848
10	2	2	27.777485	4.943848
11	3	2	28.189758	5.584717
12	3	2	29.953782	5.584717
13	2	3	28.350951	6.774902
14	2	2	28.246679	4.943848
15	2	2	27.930228	4.943848
16	3	2	28.328602	5.584717
17	2	2	28.708958	4.943848
18	2	2	27.692907	4.943848
19	2	2	28.324412	4.943848
20	2	2	29.894514	4.943848

TRITA-ICT-EX-2014:136